Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Тема работы**

**Управление потоками в ОС.**

**Обеспечение синхронизации между потоками**

Студент: Шаларь Игорь Павлович

Группа: М8О-208Б-20

Вариант: 4

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков.
6. Исходный код
7. Демонстрация работы программы
8. Выводы

**Репозиторий**

<https://github.com/IgShalar/OS/tree/main/os_lab3>

**Постановка задачи**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы.

Отсортировать массив целых чисел при помощи TimSort.

**Общие сведения о программе**

Написано для Unix.

\_$ cmake CMakeLists.txt \_$ make

\_$ ./parent

\_$ ./lab3 [максимальное количество потоков]

**main.cpp**: выполняет timsort и контролирует многопоточность.

Системные вызовы:

thread thr(&thread\_control::insertion\_sort, this, ref(v), s, l);

thr.detach();

Создание и запуск потока.

mtx.unlock();

mtx.lock();

Блокировка и освобождение mutex-а.

**mylib.cpp**: вспомогательные функции.

**Общий метод и алгоритм решения**

Тимсорт модифицирован для лучшей работы с многопоточностью.

Сначала исходная последовательность разбивается на упорядоченные или частично упорядоченные подпоследовательности не меньше вычисляемой длины. В режиме многопоточности подпоследовательности сортируются сортировкой вставкой и объединяются по аналогии с сортировкой слиянием.

**Исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков.**

Для размера последовательности 1000.

| **p** | **T1** (сек) | **Tp** (сек) | **Sp** = T1/Tp | **Xp** = Sp/p |
| --- | --- | --- | --- | --- |
| 2 | 0.00908 | 0.00766 | 1.185 | 0.593 |
| 3 | 0.00908 | 0.00599 | 1.516 | 0,505 |
| 4 | 0.00908 | 0.00552 | 1.645 | 0.411 |
| 5 | 0.00908 | 0.00565 | 1.607 | 0.3214 |
| 6 | 0.00908 | 0.00566 | 1.604 | 0.267 |

Для размера последовательности 100.

| **p** | **T1** (сек) | **Tp** (сек) | **Sp** = T1/Tp | **Xp** = Sp/p |
| --- | --- | --- | --- | --- |
| 2 | 0.00085 | 0.0007 | 1.214 | 0.607 |
| 3 | 0.00085 | 0.00078 | 1.09 | 0.363 |
| 4 | 0.00085 | 0.00076 | 1.118 | 0.2795 |
| 5 | 0.00085 | 0.00069 | 1.232 | 0.2464 |

При обработке последовательности длиной около 1000 программа редко использует более 3 потоков, а если длина около 100 - более 2.

**Исходный код**

**main.cpp**

#include<iostream>

#include<unistd.h>

#include<thread>

#include<vector>

#include<chrono>

#include"mylib.h"

using namespace std;

const int wait\_time = 100;

class thread\_control{

private:

mutex mtx;

int num;

void stop(){

mtx.lock();

num--;

mtx.unlock();

}

void insertion\_sort(vector <int> &v, int s, int l){ //Сортировка вставками подмассива

for (int i = s + 1; i < s + l; i++){

int j = i;

while(j != s && v[j] < v[j - 1]){

swap(v[j], v[j - 1]);

j--;

}

}

stop();

}

void my\_merge(vector <int> &v, pair <int, int> a, pair<int,int> b){

int s1 = a.first;

int l1 = a.second;

int s2 = b.first;

int l2 = b.second;

if (s1 > s2){

swap (s1, s2);

swap (l1, l2);

}

vector <int> t (l1);

for (int i = 0; i < l1; i++) t[i] = v[s1 + i];

int it1 = 0, it2 = s2, it3 = s1;

while (it1 < l1 && it2 < s2 + l2){

if (t[it1] < v[it2]){

v[it3] = t[it1];

it1++;

}

else{

v[it3] = v[it2];

it2++;

}

it3++;

}

while (it1 < l1){

v[it3] = t[it1];

it1++;

it3++;

}

while (it2 < s2 + l2){

v[it3] = v[it2];

it2++;

it3++;

}

stop();

}

public:

int max;

thread\_control(){

num = 0;

max = 5;

}

void start\_sort(vector <int> &v, int s, int l){

while (num == max) this\_thread::sleep\_for(std::chrono::milliseconds(wait\_time));

mtx.lock();

if (num == max){

mtx.unlock();

start\_sort(ref(v), s, l);

return;

}

thread thr(&thread\_control::insertion\_sort, this, ref(v), s, l);

thr.detach();

num++;

mtx.unlock();

}

void start\_merge(vector <int> &v, pair <int, int> a, pair<int,int> b){

while (num == max) this\_thread::sleep\_for(std::chrono::milliseconds(wait\_time));

mtx.lock();

if (num == max){

mtx.unlock();

start\_merge(ref(v), a, b);

return;

}

thread thr(&thread\_control::my\_merge, this, ref(v), a, b);

thr.detach();

num++;

mtx.unlock();

}

void wait(){

while (num != 0) this\_thread::sleep\_for(std::chrono::milliseconds(wait\_time));

}

};

thread\_control now;

int main(int num, char \* arg[]){

string arg\_get = arg[1];

now.max = stoi(arg\_get);

int n, thr\_cnt = 0, mx;

cin >> n;

vector <int> v(n);

for (int i = 0; i < n; i++) cin >> v[i];

if (n == 1){

cout << v[0] << endl;

return 0;

}

int minrun = find\_minrun(n);

vector <pair<int,int> > &q = \* divide(minrun, &v); //Входной массив разделяется на подмассивы

for (int i = 0; i < q.size(); i++) q[i] = {q[i].first, size(&q[i])};

for (int i = 0; i < q.size(); i++) now.start\_sort(ref(v), q[i].first, q[i].second);

now.wait();

while (q.size() != 1){

vector <pair <int, int> > temp;

for (int i = 1; i < q.size(); i += 2){

now.start\_merge(ref(v), q[i], q[i - 1]);

temp.push\_back(pair\_merge(q[i], q[i - 1]));

}

if (q.size() % 2 == 1) temp.push\_back(q[q.size() - 1]);

q = temp;

now.wait();

}

cout << endl;

for (int i = 0; i < v.size(); i++) cout << v[i] << " ";

cout << endl;

return 0;

}

mylib.cpp

#include<vector>

using namespace std;

int size(pair <int, int> \* a){

return a->second - a->first + 1;

}

int find\_minrun(int n){

int minrun, r = 0;;

if (n < 64) minrun = max(n / 4, 1); /\*

чтобы продемонстрировать парралельную обработку при небольших n

\*/

else{

while (n >= 64) {

r |= n & 1;

n >>= 1;

}

minrun = n + r;

}

return minrun;

}

vector <pair <int, int> > \* divide(int minrun, vector <int> \* t){

vector <pair <int, int> > \* q = new vector <pair <int, int> >;

vector <int> &v = \*t;

int n = v.size();

pair <int, int> p = {0, -1};

int now = 1;

while (now < n){

while (v[now] < v[now - 1] && now < n) now++;

if (now > p.first + 1){

reverse(v.begin() + p.first, v.begin() + now);

if (now - p.first < minrun) now = min(p.first + minrun, n);

p.second = now - 1;

q->push\_back(p);

p = {-1, -1};

p.first = now;

now++;

}

if (now == n) break;

while (v[now] >= v[now - 1] && now < n) now++;

if (now > p.first + 1){

if (now - p.first < minrun) now = min(p.first + minrun, n);

p.second = now - 1;

q->push\_back(p);

p = {-1, -1};

p.first = now;

now++;

}

}

if (p.first == n - 1 && p.second == -1) q->push\_back({n - 1, n - 1});

return q;

}

pair <int, int> pair\_merge (pair <int, int> a, pair <int, int> b){

a.first = min(a.first, b.first);

a.second += b.second;

return a;

}

mylib.h:

#ifndef MYLIB\_H

#define MYLIB\_H

#include<vector>

using namespace std;

int size (pair <int, int> \*);

int find\_minrun(int);

vector <pair <int, int> > \* divide(int, vector <int> \*);

pair <int, int> pair\_merge (pair <int, int>, pair <int, int>);

#endif

**Демонстрация работы программы**

**1)Ввод:**

./lab3 2

5

2 4 0 1 2

**Вывод:**

0 1 2 2 4

**2)Ввод:**

./lab3 3

10

1 5 3 2 4 6 8 7 3 2

**Вывод:**

1 2 2 3 3 4 5 6 7 8

**3)Ввод:**

3

1 2 3

**Вывод:**

1 2 3

**Выводы**

Освоил основы многопоточности, в т.ч. синхронизацию потоков. Научился оценивать ускорение и эффективность многопоточного алгоритма.