Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Операционные системы»
III семестр
Задание 3: «Управление потоками в ОС»

Группа:	М8О-108Б-18, №6
Студент:	Васильева Василиса Евгеньевна
Преподаватель:	Миронов Евгений Сергеевич
Оценка:	
Дата:	10.03.2020

Москва, 2019

1. Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом.
Вариант 5: Отсортировать массив строк при помощи четно-нечетной сортировки Бетчера

2. Адрес репозитория на GitHub

https://github.com/vasilisavasileva/OS_3

3. Код программы на C++

Main.cpp

```
#include<iostream>
#include<vector>
#include<Windows.h>
#include"EvenOddSort.h"
#include<limits>

int main() {

	std::cout << "Write size of vector: ";
	int n;
	std::cin >> n;
	std::vector<int> a(n);
	for (int i = 0; i < n; ++i) {
		std::cin >> a[i];
	}

	std::cout << "Write count of threads: ";
	int l;
	std::cin >> l;
	int oldSize = a.size();
	while ((double)((int)log2(a.size())) != log2(a.size())) {
		a.push_back(INT_MAX);
	}
	OddEvenMergeSort(a, l, 0, n - 1);
	for (int i = 0; i < a.size() - oldSize;++i)
		a.pop_back();
	for (int i = 0; i < n; ++i) {
```

```
            std::cout << a[i] << ' ';
        }
        std::cout << std::endl;
        system("pause");
        return 0;
    }
```

EvenOddSort.h

```
#pragma once
#include<iostream>
#include<vector>
#include<Windows.h>
#include<malloc.h>
//*
struct Param {
        std::vector<int>& a;
        int& n;
        int l;
        int r;
        Param(std::vector<int>& _a, int& _n, int _l, int _r) : a(_a), n(_n), l(_l), r(_r)
{}
};
void OddEvenMergeSort(std::vector<int>& a, int& n, int l, int r);
void shuffle(std::vector<int>& a, int l, int r)
{
        int half = (l + r) / 2;
        std::vector<int> tmp(a.size());
        int i = l, j = half + 1;
        int k = l;
        while (i <= half && j <= r) {
                if (a[i] < a[j]) {
                        tmp[k] = a[i];
                        ++i;
                        ++k;
                }
                else {
                        tmp[k] = a[j];
                        ++j;
                        ++k;
                }
        }
        for (int n = i; n <= half; ++n) {
                tmp[k] = a[n];
                ++k;
```

```
	}

	for (int n = j; n <= r; ++n) {
		tmp[k] = a[n];
		++k;
	}
	for (int i = l; i <= r; ++i) {
		a[i] = tmp[i];
	}
}


void unshuffle(std::vector<int>& a, int l, int r)
{
	int half = (l + r) / 2;
	std::vector<int> tmp(a.size());
	int i, j;
	for (i = l, j = 0; i <= r; i += 2, j++)
	{
		tmp[l + j] = a[i];
		tmp[half + j + 1] = a[i + 1];
	}
	for (int i = l; i <= r; i++)
		a[i] = tmp[i];
}

void compexch(int& a, int& b)
{
	if (b < a)
		std::swap(a, b);
}
DWORD WINAPI OddEvenMergeSortThread(LPVOID Parametr) {
	Param* P = (Param*)Parametr;
	int& r = P->r;
	int& l = P->l;
	int& n = P->n;
	std::vector<int>& a = P->a;
	if (r == l + 1)
		compexch(a[l], a[r]);
	if (r < l + 2)
		return 0;
	unshuffle(a, l, r);
	int half = (l + r) / 2;
```

```
	HANDLE Thread1 = INVALID_HANDLE_VALUE;
	HANDLE Thread2 = INVALID_HANDLE_VALUE;
	Param* P1 = nullptr;
	Param* P2 = nullptr;
	if (n != 0) {
		--n;
		P1 = new Param(a, n, l, half);
		Thread1 = CreateThread(NULL, 0, OddEvenMergeSortThread, P1, 0,
NULL);
	}
	else {
		OddEvenMergeSort(a, n, l, half);
	}
	if (n != 0) {
		--n;
		P2 = new Param(a, n, half + 1, r);
		Thread2 = CreateThread(NULL, 0, OddEvenMergeSortThread, P2, 0,
NULL);
	}
	else {
		OddEvenMergeSort(a, n, half + 1, r);
	}

	if (Thread1 != INVALID_HANDLE_VALUE) {
		WaitForSingleObject(Thread1, INFINITE);
		free(P1);
	}
	if (Thread2 != INVALID_HANDLE_VALUE) {
		WaitForSingleObject(Thread2, INFINITE);
		free(P2);
	}
	shuffle(a, l, r);
	for (auto i = l + 1; i < r; i += 2)
		compexch(a[i], a[i + 1]);
	int halfSize = (r - l + 1) / 2 - 1;
	for (int i = l + 1; i + halfSize < r; i++)
		compexch(a[i], a[i + halfSize]);
}

void OddEvenMergeSort(std::vector<int>& a, int& n, int l, int r) {
	if (r == l + 1)
		compexch(a[l], a[r]);
	if (r < l + 2)
		return;
	unshuffle(a, l, r);
```

```
	int half = (l + r) / 2;

	HANDLE Thread1 = INVALID_HANDLE_VALUE;
	HANDLE Thread2 = INVALID_HANDLE_VALUE;
	Param* P1 = nullptr;
	Param* P2 = nullptr;
	if (n != 0) {
		--n;
		P1 = new Param(a, n, l, half);
		Thread1 = CreateThread(NULL, 0, OddEvenMergeSortThread, P1, 0,
NULL);
	}
	else {
		OddEvenMergeSort(a, n, l, half);
	}
	if (n != 0) {
		--n;
		P2 = new Param(a, n, half + 1, r);
		Thread2 = CreateThread(NULL, 0, OddEvenMergeSortThread, P2, 0,
NULL);
	}
	else {
		OddEvenMergeSort(a, n, half + 1, r);
	}

	if (Thread1 != INVALID_HANDLE_VALUE) {
		WaitForSingleObject(Thread1, INFINITE);
		free(P1);
	}
	if (Thread2 != INVALID_HANDLE_VALUE) {
		WaitForSingleObject(Thread2, INFINITE);
		free(P2);
	}
	shuffle(a, l, r);
	for (int i = l + 1; i < r; i += 2)
		compexch(a[i], a[i + 1]);
	int halfSize = (r - l + 1) / 2 - 1;
	for (int i = l + 1; i + halfSize < r; i++)
		compexch(a[i], a[i + halfSize]);
}
```

4. Результаты выполнения тестов

Входные данные	Результат
Write size of vector: 4 4 99 5 3	3 4 5 99

Write count of threads: 4	
Write size of vector: 16 0 3 2 44 5 3 4 55 4 9 7 5 3 2 88 745 Write count of threads: 4	0 2 2 3 3 3 4 4 5 5 7 9 44 55 88 745
Write size of vector: 7 9 3 4 33 5 4 6 Write count of threads: 4	3 4 4 5 6 9 33

5. Объяснение результатов работы программы

Думаю, что объяснение принципа работы программы стоит начать с принципа самой четной-нечетной сортировки Бетчера. Сортировка Бетчера очень похожа на сортировку слиянием. Массив разбивается на две части и рекурсивно вызывается функция сортировки. При слиянии мы берем по первому индексу из каждой половину и продвигаемся этими своеобразными итераторами, записывая в результирующий массив меньшее из сравниваемых чисел. Т.к. обе половины уже упорядочены, все работает корректно. Данная сортировка в простом варианте работает с массивами длиной, кратной двойке. Если длина не соответствует этому условию, то вектор добивается до необходимой длины вспомогательными параметрами, которые потом будут безболезненно удалены из конца.

6. Вывод

В этой лабораторной мы используем потоки для разбиения задачи на более мелкие подзадачи. Потоки - это легковесные процессы, которые обеспечивают производительность программы за счет параллелизма внутри нее. Каждый поток относится к какому-то процессу и существует внутри него.