ФАКУЛЬТЕТ «Информатика и системы управления»  
КАФЕДРА «Информационная безопасность»

ОТЧЕТ  
по лабораторной работе № 7  
по учебной дисциплине «Алгоритмические языки»  
на тему: «Изучение потоковой многозадачности»

Вариант 3 (23)

Выполнил:   
Студент 1 курса, гр. ИУ8-24  
Спиридонов Олег

**Цель работы:**

Изучить принципы потоковой многозадачности, используя шаблон future и функцию async и класс thread.

**Условие задачи:**

Часть 1

Представить 2 варианта программы: использование шаблона future и функции async и использование класса thread.

Вариант № 3

Реализовать программу, в которой кроме главного создается три отдельных потока: первый поток сортирует первую половину вещественного массива, второй поток сортирует вторую половину вещественного массива, третий поток запускается после завершение первых двух, он сортирует массив полностью (уже частично отсортированный). Каждый поток имеет свое имя (например, thread1, thread2, thread3), и печатает отсортированный массив, перед печатью каждого значения элемента массива поток должен напечатать с новой строки свое имя. После завершения дочерних потоков главный поток выдает сообщение об окончании работы. Имена потоков и массив передаются в потоковую функцию через ее параметры, использовать одну потоковую функцию для всех трех потоков. Массив предварительно до запуска потоков заполняются числами с использованием ГПСЧ. Для сортировки использовать метод прямого выбора.

1. Запустить программу несколько раз при одних и тех же исходных данных, посмотреть, как меняются результаты вывода. Сделать выводы.

2. Между печатью имени потока и значением установить небольшую задержку, например, 10 мс. Посмотреть, как меняются результаты вывода. Сделать выводы.

Часть 2

Для своего варианта обеспечить синхронизацию потоков:

1. Обеспечить печать имени потока и значения в одну строку без возможных разрывов, продемонстрировать два варианта реализации: использование mutex и использование блокировки.

2. С помощью условной переменной обеспечить, чтобы главный поток дожидался завершения дочерних потоков (дочерние потоки перед завершением оповещают главный поток, главный поток принимает эти оповещения). Главный поток после приема оповещения от каждого дочернего потока печатает об этом событии сообщение.

**Выполнение работы:**

Файл **AL Laba 7 (multithreading - future).cpp:**

#include <iostream>

#include <vector>

#include <future>

using namespace std;

void print\_vector(const string& thread\_name, const vector<double>& array) {

for (int i = 0; i < array.size(); i++) {

cout << thread\_name << ": " << array[i] << endl;

}

cout << thread\_name << "\_total: ";

for (int i = 0; i < array.size(); i++) {

cout << array[i] << " ";

}

cout << endl;

cout << endl;

};

void direct\_selection\_sort(vector<double>& array, int start, int end, const string& thread\_name) {

for (int i = start + 1; i < end; i++) {

int placeholder = array[i];

int j = i - 1;

while (j >= start and array[j] > placeholder) {

array[j + 1] = array[j];

j--;

}

array[j + 1] = placeholder;

}

print\_vector(thread\_name, array);

};

int main() {

vector<double> array;

srand(time(0));

for (int i = 0; i < 10; i++) {

array.push\_back(rand() % 10);

}

print\_vector("main", array);

auto future1 = async(launch::async, direct\_selection\_sort, ref(array), 0, (array.size() / 2), "thread 1");

auto future2 = async(launch::async, direct\_selection\_sort, ref(array), (array.size() / 2), array.size(), "thread 2");

future1.get();

future2.get();

auto future3 = async(launch::async, direct\_selection\_sort, ref(array), 0, array.size(), "thread 3");

future3.get();

cout << "MY WORK IS DONE\n";

}

**Результат работы программы:**

main: 2

main: 2

main: 4

main: 1

main: 0

main: 5

main: 3

main: 4

main: 1

main: 3

main\_total: 2 2 4 1 0 5 3 4 1 3

thread 1: 0

thread 1: 1

thread 1: 2

thread 1: 2

thread 1: 4

thread 1: 1

thread 1: 3

thread 1: 3

thread 1: 4

thread 1: 5

thread 1\_total: 0 1 2 2 4 1 3 3 4 5

thread 2: 0

thread 2: 1

thread 2: 2

thread 2: 2

thread 2: 4

thread 2: 1

thread 2: 3

thread 2: 3

thread 2: 4

thread 2: 5

thread 2\_total: 0 1 2 2 4 1 3 3 4 5

thread 3: 0

thread 3: 1

thread 3: 1

thread 3: 2

thread 3: 2

thread 3: 3

thread 3: 3

thread 3: 4

thread 3: 4

thread 3: 5

thread 3\_total: 0 1 1 2 2 3 3 4 4 5

MY WORK IS DONE

Файл **AL Laba 7 (multithreading - threads).cpp:**

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <chrono>

#include <condition\_variable>

using namespace std;

mutex mut;

condition\_variable con\_var;

bool First\_half\_sorted = false;

bool Second\_half\_sorted = false;

void print\_vector(const string& thread\_name, const vector<double>& array) {

for (int i = 0; i < array.size(); i++) {

cout << thread\_name << ": " << array[i] << endl;

}

cout << thread\_name << "\_total: ";

for (int i = 0; i < array.size(); i++) {

cout << array[i] << " ";

}

cout << endl;

cout << endl;

};

void direct\_selection\_sort(vector<double> &array, int start, int end){

for (int i = start + 1; i < end; i++) {

int placeholder = array[i];

int j = i - 1;

while (j >= start and array[j] > placeholder) {

array[j + 1] = array[j];

j--;

}

array[j + 1] = placeholder;

}

};

void sort\_first\_half(vector<double>& array) {

//mut.lock();

direct\_selection\_sort(array, 0, (array.size() / 2));

mut.lock();

//lock\_guard <mutex> lock(mut);

print\_vector("thread1", array);

mut.unlock();

First\_half\_sorted = true;

con\_var.notify\_one();

};

void sort\_second\_half(vector<double>& array) {

//mut.lock();

direct\_selection\_sort(array, (array.size() / 2), array.size());

mut.lock();

//lock\_guard <mutex> lock(mut);

print\_vector("thread2", array);

mut.unlock();

Second\_half\_sorted = true;

con\_var.notify\_one();

};

void sort\_full(vector<double>& array) {

unique\_lock<mutex> lock(mut);

con\_var.wait(lock, [] {return First\_half\_sorted and Second\_half\_sorted;});

lock.unlock();

direct\_selection\_sort(array, 0, array.size());

print\_vector("thread3", array);

First\_half\_sorted = false;

Second\_half\_sorted = false;

};

int main() {

vector<double> array;

srand(time(0));

for (int i = 0; i < 10; i++) {

array.push\_back(rand() % 10);

}

print\_vector("main", array);

thread thread1(sort\_first\_half, ref(array));

thread thread2(sort\_second\_half, ref(array));

thread thread3(sort\_full, ref(array));

thread1.join();

thread2.join();

thread3.join();

cout << "MY WORK IS DONE\n";

}

**Результат работы программы:**

main: 4

main: 1

main: 8

main: 1

main: 9

main: 1

main: 2

main: 3

main: 6

main: 5

main\_total: 4 1 8 1 9 1 2 3 6 5

thread1: 1

thread1: 1

thread1: 4

thread1: 8

thread1: 9

thread1: 1

thread1: 2

thread1: 3

thread1: 5

thread1: 6

thread1\_total: 1 1 4 8 9 1 2 3 5 6

thread2: 1

thread2: 1

thread2: 4

thread2: 8

thread2: 9

thread2: 1

thread2: 2

thread2: 3

thread2: 5

thread2: 6

thread2\_total: 1 1 4 8 9 1 2 3 5 6

thread3: 1

thread3: 1

thread3: 1

thread3: 2

thread3: 3

thread3: 4

thread3: 5

thread3: 6

thread3: 8

thread3: 9

thread3\_total: 1 1 1 2 3 4 5 6 8 9

MY WORK IS DONE

**Вывод:**

В ходе выполнения лабораторной работы были изучены принципы потоковой многозадачности. Для выполнения задачи были реализованы 2 варианта программы: один с использованием шаблона future и функции async, другой с использованием класса thread.