ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

ДОМАШНЕЕ ЗАДАНИЕ №4

по дисциплине «Архитектура вычислительных систем»

Вариант 9

Отчет

Исполнитель:
студент 194 группы 2 подгруппы
Е. А. Ермаченко
22 ноября 2020 г.

Москва 2020

СОДЕРЖАНИЕ

1. Текст задания

Определить, является ли множество C объединением множеств A и B (A∪B), пересечением множеств (A∩B), разностью множеств A и B (A\B), разностью множеств B и A (B\A). Входные данные: множества целых положительных чисел A, B, C. Оптимальное количество потоков выбрать самостоятельно.

2. Модель параллельного программирования «Управляющий и рабочие».

В модели делегирования один поток («управляющий») создает потоки («рабочие») и назначает каждому из них задачу. Управляющему потоку нужно ожидать до тех пор, пока все потоки не завершат выполнение своих задач. Управляющий поток делегирует задачу, которую каждый рабочий поток должен выполнить, путем задания некоторой функции. Вместе с задачей на рабочий поток возлагается и ответственность за ее выполнение и получение результатов. Кроме того, на этапе получения результатов возможна синхронизация действий с управляющим (или другим) потоком.

Управляющий поток может создавать рабочие потоки в результате запросов, обращенных к системе. При этом обработка запроса каждого типа может быть делегирована рабочему потоку. В этом случае управляющий поток выполняет некоторый цикл событий. По мере возникновения событий рабочие потоки создаются и на них тут же возлагаются определенные обязанности. Для каждого нового запроса, обращенного к системе, создается новый поток. При использовании такого подхода процесс может превысить предельный объем выделенных ему ресурсов или предельное количество потоков. В качестве альтернативного варианта управляющий поток может создать пул потоков, которым будут переназначаться новые запросы. Управляющий поток создает во время инициализации некоторое количество потоков, а затем каждый поток приостанавливается до тех пор, пока не будет добавлен запрос в их очередь. По мере размещения запросов в очереди управляющий поток сигнализирует рабочему о необходимости обработки запроса. Как только поток справится со своей задачей, он извлекает из очереди следующий запрос. Если в очереди больше нет доступных запросов, поток приостанавливается до тех пор. пока управляющий поток не просигналит ему о появлении очередного задания в очереди. Если все рабочие потоки должны разделять одну очередь, то их можно программировать на обработку запросов только определенного типа. Если тип запроса в очереди не совпадает с типом запросов, на обработку которых ориентирован данный поток, то он может снова приостановиться. Главная цель управляю-потока — создать все потоки, поместить задания в очередь и «разбудить» рабочие потоки, когда эти задания станут доступными. Рабочие потоки справляются о наличии запроса в очереди, выполняют назначенную задачу и приостанавливаются сами, если для них больше нет работы. Все рабочие и управляющий потоки выполняются параллельно.[1]

3. Реализация программы

После считывания множеств формируются массивы с функциями над множествами (объединение множеств A и B, пересечение множеств A и B, разность множеств A и B или разность множеств B и A) и именами функций этих множеств. Затем в основном (управляющем) потоке посредством OpenMP[1] создаются 4 потока (рабочие). Каждый поток по своему номеру обращается к нужной функции из массива и выполняет ее. Далее задается критическая секция, где поток записывает результат функции.

[1] **OpenMP** (Open Multi-Processing) — открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран.

4. Тестирование программы

Ввод и вывод в программе реализован с помощью файлов.

Запуск программы осуществляется с помощью командной строки. При вызове программы в командной строке передаются 2 аргумента: путь к файлу с входными данными и путь к файлу с выходными данными.

На вход программе подается файл с информацией о множествах A, B и C, состоящих из целых положительных чисел. Первая, третья и пятая строки содержат размер соответствующих множеств, а вторая, четвертая и шестая строки содержат сами множества. Гарантируется, что все элементы в множествах различны и количество элементов множества соответствует его размеру.

Все примеры можно посмотреть в папке **examples**. В папке **examples/input** файлы test[номер].txt содержат входные данные для программы, а в папке **examples/output** файлы answer[номер].txt содержат результат выполнения программы для тестов с соответствующим номером.

Список использованной литературы

1) Хьюз Камерон. Параллельное и распределенное программирование на C++/ Хьюз Камерон, Хьюз Трейси; пер. Н. М. Ручко. – Москва: Вильямс 2004. – 130 с.

2) Карепова Е. Д., Кузьмин Д. А., Легалов А.И., Редькин А. В., Удалова Ю. В., Федоров Г. А. Средства разработки параллельных программ. Учебное пособие. Красноярск 2007

ПРИЛОЖЕНИЕ

Код программы:

```
#include <set>
#include <omp.h>
#include <algorithm>
#include<fstream>

// краткие имена типов для удобства
using set = typename std::set<size_t>;
using It = typename std::set<size_t>::iterator;
// итератор для заполнения результата функции
using Inserter = std::insert_iterator<set>;
// тип функции для вычесления множества(пересечение, объединение или разность)
typedef Inserter(*Function)(It, It, It, It, Inserter);

/// <summary>
/// Считываем множества из файла
/// </summary>
/// <param name="path">путь к файлу</param>
/// <param name="A">множество A</param>
/// <param name="B">множество B</param>
/// <param name="C">множество C</param>
void read_sets(std::string path, set& A, set& B, set& C)
{
	std::ifstream in(path);

	if (in.is_open())
	{
		size_t n;
		in >> n;
		for (size_t i = 0; i < n; i++)
		{
			int el;
			in >> el;
			A.insert(el);
		}

		in >> n;
		for (size_t i = 0; i < n; i++)
		{
			int el;
			in >> el;
			B.insert(el);
		}

		in >> n;
		for (size_t i = 0; i < n; i++)
		{
			int el;
			in >> el;
			C.insert(el);
		}
	}
}
/// <summary>
/// Пишем множество в файл
/// </summary>
/// <param name="set">множество для записи</param>
/// <param name="out">поток, в который пишется множество</param>
void print_set(set set, std::ofstream& out)
{
```

```
	out << "{ ";
	for (auto i = set.begin(); i != set.end(); i++)
	{
		out << *i << " ";
	}

	out << "}" << std::endl;
	out << std::endl;
}



int main(int argc, char* argv[])
{
	// открываем файл на запись
	std::ofstream out;
	out.open(argv[2]);
	size_t const threads_num = 4;
	set A;
	set B;
	set C;
	// инициализируем множества
	read_sets(argv[1], A, B, C);
	// пишем полученные множества в файл
	out << "A = ";
	print_set(A, out);
	out << "B = ";
	print_set(B, out);
	out << "C = ";
	print_set(C, out);
	// функции операций над множествами
	Function functions[threads_num];
	functions[0] = std::set_union<It, It, Inserter>;
	functions[1] = std::set_intersection<It, It, Inserter>;
	functions[2] = std::set_difference<It, It, Inserter>;
	functions[3] = std::set_difference<It, It, Inserter>;
	// имена операций над множествами
	std::string names[threads_num];
	names[0] = "union of A and B";
	names[1] = "intersection of A and B";
	names[2] = "difference between A and B";
	names[3] = "difference between B and A";
	// задаем 4 потока
#pragma omp parallel num_threads(threads_num)
	{
		// номер потока
		auto num = omp_get_thread_num();

		set result;
		// если нужно посчитать разность B и A, то меняем аргументы местами
		if (num == 3)
			functions[num](B.begin(), B.end(), A.begin(), A.end(),
std::inserter(result, result.begin()));
		else
			functions[num](A.begin(), A.end(), B.begin(), B.end(),
std::inserter(result, result.begin()));
		// критическая секция для записи
#pragma omp critical
		{
			if (C == result)
			{
				out << "C is " << names[num] << std::endl;
			}
			else
```

```
                        {
                                out << "C is not " << names[num] << std::endl;
                        }
                        out << names[num] << " = ";
                        print_set(result, out);
                }
        }

        return 0;
}
```