Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Лабораторная работа № 7

ПО КУРСУ

«Алгоритмические языки»

на тему «Потоковая многозадачность»

Студент ИУ8-23 (Группа) В. С. Ажгирей (И. О. Фамилия)

Преподаватель: М. В. Малахов (И.О. Фамилия)

2024

Введение

Цели и задачи работы

Цель работы состоит в овладении навыками разработки многопоточных программ на языке Си++, Для достижения цели необходимо выполнить следующие задачи:

- изучить необходимые учебные материалы, посвященные поточной многозадачности языка Си++;

- разработать программы на языке Си++ для решения заданных вариантов заданий;

- отладить программы;

- выполнить решение контрольного примера небольшой размерности с помощью программы и ручной расчет контрольного примера;

- подготовить отчет по лабораторной работе.

Условия для 1 варианта

Часть 1: Реализовать программу, в которой кроме главного создается три отдельных потока: первый поток печатает n1 первых значений последовательности Фибоначчи; второй поток печатает n2 значений первых натуральных чисел (1, 2, 3, … n2), третий поток печатает n3 значений, выдаваемых ГПСЧ. Значения n1, n2, n3 задаются константами или вводятся с клавиатуры (примерный диапазон 20..30). Каждый поток имеет свое имя (например, thread1, thread2, thread3), перед печатью значения, поток должен напечатать с новой строки свое имя. После завершения дочерних потоков главный поток выдает сообщение об окончании работы. Имена потоков и значения n1, n2, n3 передаются в потоковую функцию через ее параметры.

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.

Основная часть

Исходный текст программы:

```
#include <thread>
#include <mutex>
#include <condition_variable>
#include <iostream>
#include <vector>
#include <chrono>


std::mutex mtx;
std::condition_variable condition;
std::vector<std::string> completed_threads;

void fibonacci(size_t f1, size_t f2, size_t i, size_t n, std::string thread_name)
{
	if (i <= n)
	{
		std::unique_lock<std::mutex> ul(mtx);
		std::cout << "thread: " << thread_name << "\t id: " <<
std::this_thread::get_id() << "\t value: " << f1 << std::endl;
		ul.unlock();
		std::this_thread::sleep_for(std::chrono::milliseconds(10));
		fibonacci(f2, f1 + f2, i + 1, n, thread_name);
	}
	else
	{
		std::unique_lock<std::mutex> cmpl(mtx);

		completed_threads.push_back(thread_name);
		condition.notify_one();
	}
}

void natural_numbers(unsigned n, std::string thread_name)
{
	for (unsigned i = 1; i < n + 1; ++i)
	{
		std::this_thread::sleep_for(std::chrono::milliseconds(10));
		std::unique_lock<std::mutex> ul(mtx);
		std::cout << "thread: " << thread_name << "\t id: " <<
std::this_thread::get_id() << "\t value: " << i << std::endl;
	}
	std::unique_lock<std::mutex> cmpl(mtx);
	completed_threads.push_back(thread_name);
	condition.notify_one();
}

void random_numbers(unsigned n, std::string thread_name)
{
	srand(time(0));
	for (unsigned i = 0; i < n; ++i)
	{
		std::this_thread::sleep_for(std::chrono::milliseconds(10));
		std::unique_lock<std::mutex> ul(mtx);
		std::cout << "thread: " << thread_name << "\t id: " <<
std::this_thread::get_id() << "\t value: " << rand() % 100 << std::endl;
	}
	std::unique_lock<std::mutex> cmpl(mtx);
	completed_threads.push_back(thread_name);
	condition.notify_one();
}
```

```
void thread_complete()
{
	size_t count_completed_threads = 0;
	while (count_completed_threads != 3)
	{
		std::unique_lock<std::mutex> cmpl(mtx);
		condition.wait(cmpl);
		std::cout << "====================== completed " <<
completed_threads[count_completed_threads] << " ======================" << std::endl;
		count_completed_threads++;
	}
}

int main()
{

	auto start = std::chrono::high_resolution_clock::now();

	srand(time(0));

	int n1, n2, n3;
	std::cout << "Enter n1, n2, n3: ";
	std::cin >> n1 >> n2 >> n3;

	std::thread thread1(fibonacci, 1, 1, 1, n1,  "thread1");
	std::thread thread2(natural_numbers, n2, "thread2");
	std::thread thread3(random_numbers, n3, "thread3");
	std::thread thread4(thread_complete);

	thread1.detach();
	thread2.detach();
	thread3.detach();
	thread4.join();

	auto end = std::chrono::high_resolution_clock::now();
	std::chrono::duration<double> duration = end - start;
	std::cout << "================ Child threads are completed ==================="
<< std::endl;
	std::cout << "Execution time: " << duration.count() << " seconds" <<
std::endl;

	return 0;
}
```

Снимки выполнения работы программы

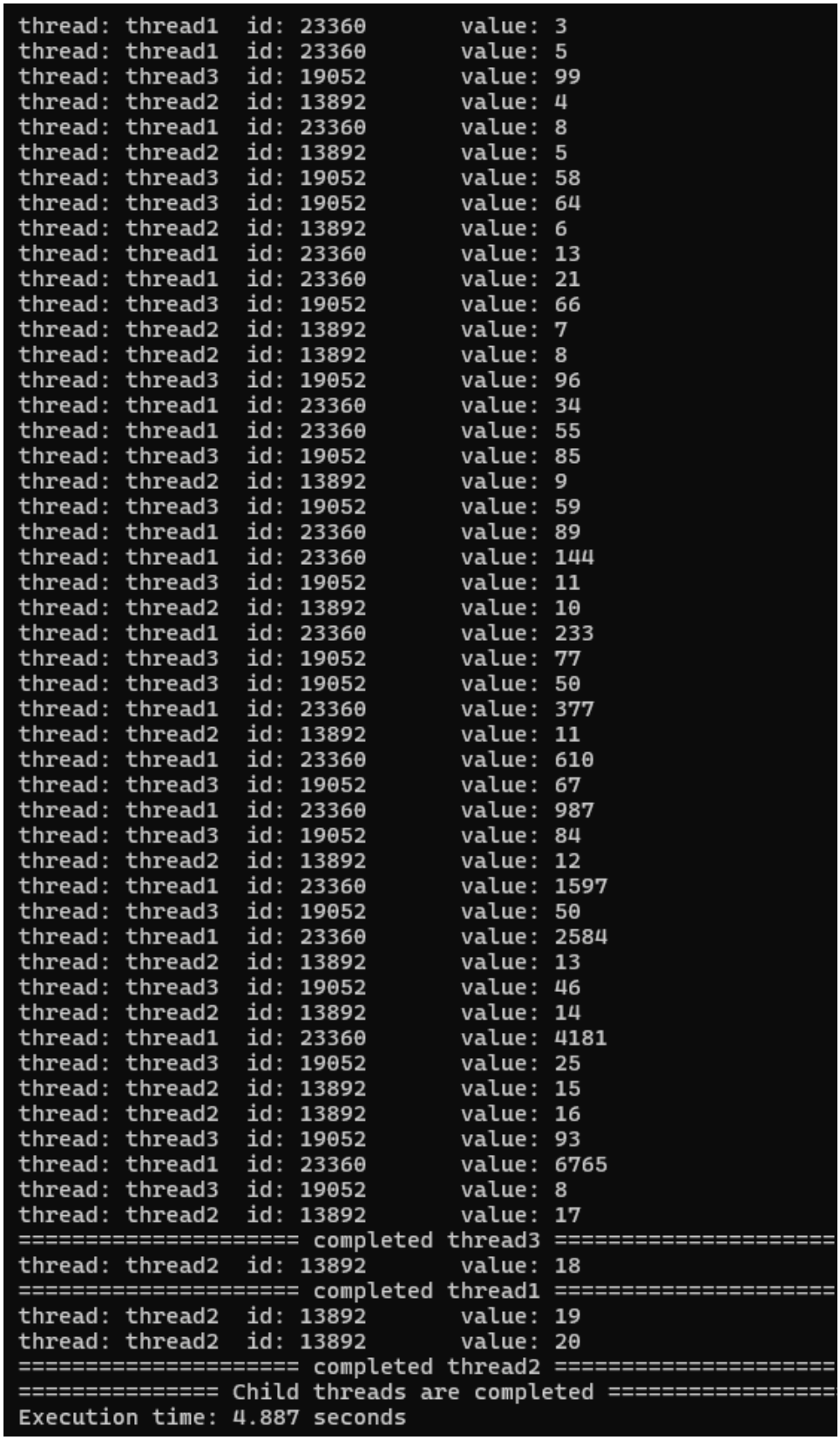

Рисунок 1 – Запуск программы с использование многопоточности

Заключение

Задачи лабораторной работы были решены, результаты проверены. Изучена на практике потоковая многозадачность в языке C/C++, методы синхронизации, блокировки и условные переменные condition_variable.