

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения

вычислительной техники и автоматизированных
систем

Лабораторная работа №6

по дисциплине: ООП

тема: «Потоки в C++»

Выполнил: студент группы
ПВ-233
Мороз Роман Алексеевич

Проверили:
Морозов Данила Александрович

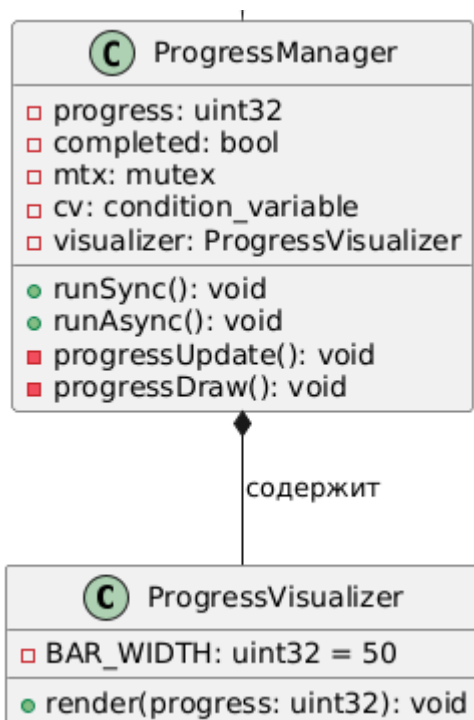
Белгород 2025

Потоки в C++

Цель работы: изучение основных возможностей потоков управления и потоков ввода-вывода. Получение навыков работы со стандартными средствами управления потоками в C++11. Знакомство с классом Thread и стандартными средствами синхронизации потоков.

Задание:

1. Изучить основные классы и их возможности работы с потоками в C++11.
 2. Разработать программу в соответствии с вариантом задания. Программа должна содержать 2 потока Thread для реализации основного задания лабораторной работы. Вывод организовать в отдельном потоке.
 3. Реализовать классы и выполнить перегрузку оператора функтора для реализации поставленной основной задачи.
 4. Разработать программу в соответствии с вариантом задания (номер варианта + 3), используя API CreateThread.
 5. Сделать выводы о проделанной работе.
-
9. Один поток выводит “визуальный progress bar” в консоль, другой поток выполняет цифровой подсчет текущего значения progress bar. Произвести синхронный вывод при каждой итерации. Показать выполнение работы программы в синхронном и асинхронном режимах.



```
#include <iostream>
```

```

#include <thread>
#include <chrono>
#include <mutex>
#include <string>
#include <condition_variable>

#define COLOR_BAR "\033[34m"
#define COLOR_TEXT "\033[33m"
#define COLOR_RESET "\033[0m"

class ProgressVisualizer {
private:
    static const u_int32_t BAR_WIDTH = 50;

public:
    void render(u_int32_t progress) const {
        std::string bar = COLOR_BAR;
        _Float32 percentage = progress / 100.0f;

        u_int32_t filled = static_cast<u_int32_t>(BAR_WIDTH *
percentage);

        bar.append(filled, '#');
        bar.append(BAR_WIDTH - filled, ' ');

        std::cout << "\r[" << bar << "]" << COLOR_TEXT << progress <<
"%"
                << COLOR_RESET << std::flush;
    }
};

class ProgressManager {
private:
    u_int32_t progress = 0;
    bool completed = false;
    std::mutex mtx;
    std::condition_variable cv;
    ProgressVisualizer visualizer;

    void progressUpdate() {
        for(u_int32_t i = 0; i <= 100; ++i) {
            {
                std::lock_guard<std::mutex> lock(mtx);

```

```

        progress = i;
    }

    cv.notify_one();

std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    {
        std::lock_guard<std::mutex> lock(mtx);
        completed = true;
    }

    cv.notify_one();
}

inline void progressDraw() noexcept {
    while(true) {
        std::unique_lock<std::mutex> lock(mtx);
        cv.wait(lock, [this] { return progress > 0 || completed;
});

        visualizer.render(progress);

        if(completed || progress >= 100) {
            visualizer.render(100);
            break;
        }
    }
}

public:
    inline void runSync() noexcept {
        for(int i = 0; i <= 100; ++i) {
            visualizer.render(i);

std::this_thread::sleep_for(std::chrono::milliseconds(100));
        }

        std::cout << "\n\n" << COLOR_TEXT << "Synchronous mode
completed!"
                << COLOR_RESET << "\n";
    }
}

```

```

inline void runAsync() noexcept {
    std::thread updater(&ProgressManager::progressUpdate, this);
    std::thread drawer(&ProgressManager::progressDraw, this);

    updater.join();
    drawer.join();

    std::cout << "\n\n" << COLOR_TEXT << "Asynchronous mode
completed!"
                << COLOR_RESET << "\n";
}
};

int main() {
    ProgressManager manager;

    std::cout << "\n" << COLOR_TEXT << "Starting synchronous mode..."
                << COLOR_RESET << "\n";

    manager.runSync();

    std::cout << "\n" << COLOR_TEXT << "Starting asynchronous mode..."
                << COLOR_RESET << "\n";

    manager.runAsync();

    return 0;
}

```

```

Asynchronous mode completed!
> g++ -std=c++17 -pthread threads.cpp -o progress
> ./progress

Starting synchronous mode...
[#####] 100%

Synchronous mode completed!

Starting asynchronous mode...
[#####] 100%

Asynchronous mode completed!

```

2. Один поток добавляет случайные символы в строку, а другой поток удаляет случайный символ из строки. Произвести синхронный вывод при каждой итерации. Показать выполнение работы программы в синхронном и асинхронном режимах.

```

#include <windows.h>
#include <iostream>
#include <vector>
#include <memory>
#include <stdexcept>
#include <cmath>

```

```

class EquationSolver {
public:
    virtual std::vector<double> solve() = 0;
    virtual void print_results() = 0;
    virtual ~EquationSolver() = default;
};

```

```

class NoRealRootsException : public std::runtime_error {
public:
    NoRealRootsException() : std::runtime_error("Уравнение не имеет действительных корней") {}
};

```

```

class QuadraticEquationSolver : public EquationSolver {
    double a, b, c;
    std::vector<double> roots;

```

```
public:
```

```
    QuadraticEquationSolver(double a, double b, double c) : a(a), b(b), c(c) {}
```

```
    std::vector<double> solve() override {  
        double discriminant = b * b - 4 * a * c;
```

```
        if (discriminant < 0) {  
            throw NoRealRootsException();  
        }
```

```
        double sqrt_d = sqrt(discriminant);  
        roots = {  
            (-b + sqrt_d) / (2 * a),  
            (-b - sqrt_d) / (2 * a)  
        };
```

```
        return roots;  
    }
```

```
    void print_results() override {  
        if (roots.empty()) {  
            std::cout << "Уравнение еще не решено" << std::endl;  
            return;  
        }
```

```
        std::cout << "Корни уравнения: ";
```

```
        for (auto root : roots) {  
            std::cout << root << " ";  
        }
```

```
        std::cout << std::endl;  
    }  
};
```

```
struct ThreadData {  
    std::shared_ptr<QuadraticEquationSolver> solver;  
    std::string error_message;  
};
```

```
DWORD WINAPI SolveEquationThread(LPVOID lpParam) {  
    ThreadData* data = static_cast<ThreadData*>(lpParam);
```

```

    try {
        data->solver->solve();
    } catch (const std::exception& ex) {
        data->error_message = ex.what();
    }

    return 0;
}

int main() {
    setlocale(0, "RUS");

    std::vector<std::shared_ptr<QuadraticEquationSolver>> solvers = {
        std::make_shared<QuadraticEquationSolver>(1, -5, 6),
        std::make_shared<QuadraticEquationSolver>(1, 2, 5),
        std::make_shared<QuadraticEquationSolver>(1, -6, 9)
    };

    std::vector<ThreadData> equations;
    for (const auto& solver : solvers) {
        equations.push_back({ solver, "" });
    }

    std::vector<HANDLE> threads;
    for (auto& eq : equations) {
        HANDLE thread = CreateThread(NULL, 0, SolveEquationThread, &eq,
0, NULL);

        if (thread) {
            threads.push_back(thread);
        }
    }

    WaitForMultipleObjects(threads.size(), threads.data(), TRUE, INFINITE);

    for (size_t i = 0; i < equations.size(); ++i) {
        std::cout << "Уравнение " << i + 1 << ": ";

        if (!equations[i].error_message.empty()) {
            std::cout << equations[i].error_message << std::endl;
        } else {
            equations[i].solver->print_results();
        }
    }
}

```



```
    }  
}
```

```
    for (auto& thread : threads) {  
        CloseHandle(thread);  
    }
```

```
    return 0;  
}
```

```
Уравнение 1: Корни уравнения: 3 2  
Уравнение 2: Уравнение не имеет действительных корней  
Уравнение 3: Корни уравнения: 3 3
```

Вывод: изучили основные возможности потоков управления и потоков ввода-вывода. Получили навыки работы со стандартными средствами управления потоками в C++11. Познакомились с классом Thread и стандартными средствами синхронизации потоков.