МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ» ИНСТИТУТ ПЕРСПЕКТИВНОЙ ИНЖЕНЕРИИ КАФЕДРА ПРИКЛАДНОЙ ИНФОРМАТИКИ

**Лабораторная работа № 1**

по дисциплине

«Многопоточное программирование»

**Выполнил:**

Мухамеджанов Артур Тимурович

Студент 3 курса

группы ПИН-б-о-22-1

Направления подготовки

09.03.03 Прикладная информатика

очной формы обучения

**Работу принял**

Мартыновская А. С.

(ФИО)

**Тема:** Реализация потоков в C++

**Задание 1:** Создание потока

**Задание 2:** Ожидание потока

**Задание 3:** Параметры потока

**Задание 4:** Принудительное завершение потока

**Задание 5:** Обработка завершения потока

**Ход работы:**

1. Напишите программу, которая создает поток. Используйте атрибуты по умолчанию. Родительский и вновь созданный поток должны распечатать десять строк текста.

#include <iostream>

#include <pthread.h>

void \* threadFunction(void \* arg) {

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

std::cout << "Новый поток: строка " << i + 1 << std::endl;

}

return nullptr;

}

int main() {

pthread\_t thread;

// Создаём поток с атрибутами по умолчанию (nullptr в качестве атрибутов)

if (pthread\_create( & thread, nullptr, threadFunction, nullptr) != 0) {

std::cerr << "Ошибка при создании потока!" << std::endl;

return 1;

}

// Родительский поток выводит свои строки

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

std::cout << "Родительский поток: строка " << i + 1 << std::endl;

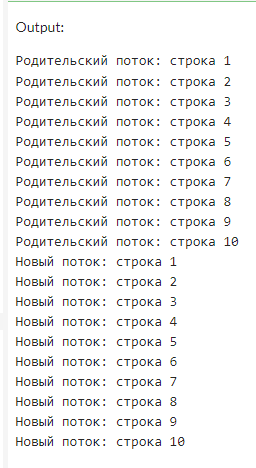
}

// Ожидаем завершения потока (можно убрать, если не нужно по условию)

pthread\_join(thread, nullptr);

return 0;

}



1. Модифицируйте программу упр. 1 так, чтобы вывод родительского потока производился после завершения дочернего. Используйте pthread\_join.

#include <iostream>

#include <pthread.h>

// Функция, выполняемая дочерним потоком

void \* threadFunction(void \* arg) {

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

std::cout << "Дочерний поток: строка " << i + 1 << std::endl;

}

return nullptr;

}

int main() {

pthread\_t thread;

// Создаём поток с атрибутами по умолчанию

if(pthread\_create( & thread, nullptr, threadFunction, nullptr) != 0) {

std::cerr << "Ошибка при создании потока!" << std::endl;

return 1;

}

// Ждём завершения дочернего потока

pthread\_join(thread, nullptr);

// После завершения дочернего потока выводим строки родительского

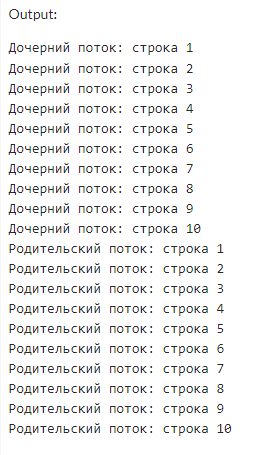
for(int i = 0; i < 10; ++i) {

std::cout << "Родительский поток: строка " << i + 1 << std::endl;

}

return 0;

}



1. Напишите программу, которая создает четыре потока, исполняющие одну и ту же функцию. Эта функция должна распечатать последовательность текстовых строк, переданных как параметр. Каждый из созданных потоков должен распечатать различные последовательности строк.

#include <iostream>

#include <pthread.h>

// Структура для передачи параметров в поток

struct ThreadData {

const char \* lines[10]; // Массив строк (до 10 строк на поток)

int lineCount; // Количество строк для вывода

};

// Функция, выполняемая потоками

void \* threadFunction(void \* arg) {

ThreadData \* data = static\_cast < ThreadData \* > (arg); // Приводим void\* к структуре

for(int i = 0; i < data -> lineCount; ++i) {

std::cout << "Поток: " << data -> lines[i] << std::endl;

}

return nullptr;

}

int main() {

const int NUM\_THREADS = 4;

pthread\_t threads[NUM\_THREADS];

ThreadData threadData[NUM\_THREADS];

// Подготовка уникальных последовательностей строк для каждого потока

threadData[0] = {{"Первая строка потока 1", "Вторая строка потока 1"}, 2};

threadData[1] = {{"Сообщение от потока 2", "Ещё одно от потока 2", "Третье от потока 2"}, 3};

threadData[2] = {{"Поток 3 приветствует", "Поток 3 прощается"}, 2};

threadData[3] = {{"Единственная строка потока 4"},1};

// Создание четырёх потоков

for(int i = 0; i < NUM\_THREADS; ++i) {

if(pthread\_create( & threads[i], nullptr, threadFunction, & threadData[i]) != 0) {

std::cerr << "Ошибка при создании потока " << i << "!" << std::endl;

return 1;

}

}

// Ожидание завершения всех потоков

for(int i = 0; i < NUM\_THREADS; ++i) {

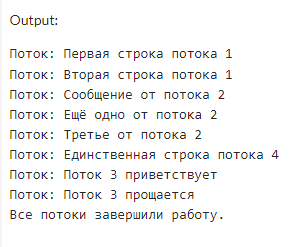
pthread\_join(threads[i], nullptr);

}

std::cout << "Все потоки завершили работу." << std::endl;

return 0;

}



1. Дочерний поток должен распечатывать текст на экран. Через две секунды после создания дочернего потока, родительский поток должен прервать ее вызовом функции pthread\_cancel.

#include <iostream>

#include <pthread.h>

#include <unistd.h> // Для sleep()

// Функция, выполняемая дочерним потоком

void \* threadFunction(void \* arg) {

int count = 1;

while(true) { // Бесконечный цикл для вывода текста

std::cout << "Дочерний поток: строка " << count++ << std::endl;

sleep(1); // Задержка 1 секунда между выводами

}

return nullptr; // Никогда не достигнется из-за бесконечного цикла

}

int main() {

pthread\_t thread;

// Создаём дочерний поток

if(pthread\_create( & thread, nullptr, threadFunction, nullptr) != 0) {

std::cerr << "Ошибка при создании потока!" << std::endl;

return 1;

}

// Даём дочернему потоку поработать 2 секунды

sleep(2);

// Прерываем дочерний поток

if(pthread\_cancel(thread) != 0) {

std::cerr << "Ошибка при отмене потока!" << std::endl;

return 1;

}

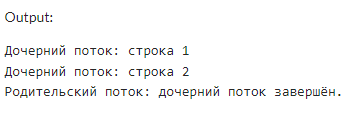
// Ждём завершения дочернего потока

pthread\_join(thread, nullptr);

std::cout << "Родительский поток: дочерний поток завершён." << std::endl;

return 0;

}



1. Модифицируйте программу упр. 4 так, чтобы дочерний поток перед завершением распечатывал сообщение об этом. Используйте pthread\_cleanup\_push.

#include <iostream>

#include <pthread.h>

#include <unistd.h> // Для sleep()

// Функция очистки, вызываемая при завершении потока

void cleanupHandler(void \* arg) {

std::cout << "Дочерний поток: завершение работы!" << std::endl;

}

// Функция, выполняемая дочерним потоком

void \* threadFunction(void \* arg) {

int count = 1;

// Регистрируем обработчик очистки

pthread\_cleanup\_push(cleanupHandler, nullptr);

while(true) { // Бесконечный цикл для вывода текста

std::cout << "Дочерний поток: строка " << count++ << std::endl;

sleep(1); // Задержка 1 секунда между выводами

}

// Удаляем обработчик (никогда не выполнится из-за бесконечного цикла)

pthread\_cleanup\_pop(0);

return nullptr; // Никогда не достигнется

}

int main() {

pthread\_t thread;

// Создаём дочерний поток

if(pthread\_create( & thread, nullptr, threadFunction, nullptr) != 0) {

std::cerr << "Ошибка при создании потока!" << std::endl;

return 1;

}

// Даём дочернему потоку поработать 2 секунды

sleep(2);

// Прерываем дочерний поток

if(pthread\_cancel(thread) != 0) {

std::cerr << "Ошибка при отмене потока!" << std::endl;

return 1;

}

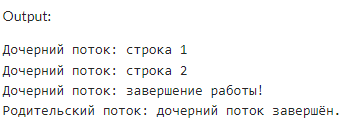
// Ждём завершения дочернего потока

pthread\_join(thread, nullptr);

std::cout << "Родительский поток: дочерний поток завершён." << std::endl;

return 0;

}



**Вывод**: Потоки являются удобным унифицированным программным интерфейсом для чтения или записи файлов, сокетов и передачи данных между процессами.

**Вопросы**:

1. Конкурентность в компьютерных системах.

2. Подходы к конкурентности.

3. Сравнение конкурентности и параллелизма.

4. Конкурентность для разделения неотложных задач.

5. Конкурентность для повышения производительности: параллелизм задач и данных.

6. Конкурентность и многопоточность в C++. История поддержки многопоточности в C++.

7. Поддержка конкурентности в стандарте C++11.

8. Расширение поддержки конкурентности и параллелизма в C++14.

9. Расширение поддержки конкурентности и параллелизма в C++17.

10. Расширение поддержки конкурентности и параллелизма в C++20.

11. Расширение поддержки конкурентности и параллелизма в C++23.

12. Эффективность, обеспеченная библиотекой потоков C++.

13. Средства, ориентированные на использование конкретной платформы.

**Ответы**:

1. **Конкурентность в компьютерных системах** — это способность системы выполнять несколько задач одновременно, переключаясь между ними или обрабатывая их параллельно.
2. **Подходы к конкурентности** — многопоточность, асинхронное программирование, событийно-ориентированный подход, акторы и кооперативная многозадачность.
3. **Сравнение конкурентности и параллелизма** — конкурентность: управление множеством задач в рамках одной системы; параллелизм: одновременное выполнение задач на нескольких процессорах.
4. **Конкурентность для разделения неотложных задач** — позволяет приоритизировать срочные задачи, распределяя ресурсы между ними и менее критичными.
5. **Конкурентность для повышения производительности** — параллелизм задач (разделение задач на независимые потоки) и данных (обработка разных частей данных одновременно).
6. **Конкурентность и многопоточность в C++** — до C++11 поддержка была через библиотеки (pthread, WinAPI), затем добавлена встроенная многопоточность в стандарт.
7. **Поддержка конкурентности в C++11** — введены std::thread, std::mutex, std::condition\_variable, std::future для управления потоками и синхронизации.
8. **C++14** — улучшены средства конкурентности: std::shared\_timed\_mutex, уточнены семантики атомарных операций.
9. **C++17** — добавлены параллельные алгоритмы STL (например, std::sort с std::execution::par), улучшена работа с std::future.
10. **C++20** — введены корутины, std::jthread, семафоры, барьеры и атомарные ожидания (std::atomic::wait).
11. **C++23** — расширены параллельные алгоритмы, улучшена поддержка корутин и добавлены новые средства синхронизации (на стадии финализации).
12. **Эффективность библиотеки потоков C++** — высокая за счёт низкоуровневого контроля и минимальных накладных расходов, но зависит от платформы и реализации.
13. **Средства для конкретной платформы** — POSIX threads (Linux), WinAPI (Windows), Grand Central Dispatch (macOS) — используются для оптимизации под конкретные ОС.