

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное учреждение высшего образования  
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

старший преподаватель		Поляк М.Д.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3

Синхронизация потоков средствами WinAPI

по дисциплине: Операционные системы

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	Z1432K		Куралесов Ф.А.
	номер группы	подпись, дата	инициалы, фамилия

Студенческий билет №	2021/3806
----------------------	-----------

Санкт-Петербург 2025

Цель работы: Знакомство с многопоточным программированием и методами синхронизации потоков средствами Windows API.

Задание на лабораторную работу:

1. С помощью таблицы вариантов заданий выбрать граф запуска потоков в соответствии с номером варианта. Вершины графа являются точками запуска/завершения потоков, дугами обозначены сами потоки. Длину дуги следует интерпретировать как ориентировочное время выполнения потока. В процессе своей работы каждый поток должен в цикле выполнять два действия:

- i. выводить букву имени потока в консоль;
- ii. вызывать функцию `computation()` для выполнения вычислений, требующих задействования ЦП на длительное время. Эта функция уже написана и подключается из заголовочного файла `lab3.h`, изменять ее не следует.

2. В соответствии с вариантом выделить на графе две группы с выполняющимися параллельно потоками. В первой группе потоки не синхронизированы, параллельное выполнение входящих в группу потоков происходит за счет планировщика задач. Вторая группа синхронизирована семафорами и потоки внутри группы выполняются в строго зафиксированном порядке: входящий в группу поток передает управление другому потоку после каждой итерации цикла (см. задачу производителя и потребителя). Таким образом потоки во второй группе выполняются в строгой очередности.

3. С использованием средств Windows API реализовать программу для последовательно-параллельного выполнения потоков в ОС Windows. Запрещается использовать какие-либо библиотеки и модули, решающие задачу кроссплатформенной разработки многопоточных приложений (`std::thread`, `Qt Thread`, `Boost Thread` и т.п.), а также функции приостановки

выполнения программы (например, `Sleep()`, `SwitchToThread()` и подобные).

Для этого необходимо написать код в файле `lab3.cpp`:

- i. Функция `unsigned int lab3_thread_graph_id()` должна возвращать номер графа запуска потоков, полученный из таблицы вариантов заданий.
- ii. Функция `const char* lab3_unsynchronized_threads()` должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно без синхронизации.
- iii. Функция `const char* lab3_sequential_threads()` должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно в строгой очередности друг за другом.
- iv. Функция `int lab3_init()` заменяет собой функцию `main()`. В ней необходимо реализовать запуск потоков, инициализацию вспомогательных переменных (мьютексов, семафоров и т.п.). Перед выходом из функции `lab3_init()` необходимо убедиться, что все запущенные потоки завершились. Возвращаемое значение: 0 - работа функции завершилась успешно, любое другое числовое значение - при выполнении функции произошла критическая ошибка.
- v. Добавить любые другие необходимые для работы программы функции, переменные и подключаемые файлы.
- vi. Создавать функцию `main()` не нужно. В проекте уже имеется готовая функция `main()`, изменять ее нельзя. Она выполняет единственное действие: вызывает функцию `lab3_init()`.
- vii. Не следует изменять какие-либо файлы, кроме `lab3.cpp`. Также не следует создавать новые файлы и писать в них код, поскольку код из этих файлов не будет использоваться во время тестирования.

4. Подготовить отчет о выполнении лабораторной работы и загрузить его под именем `report.pdf` в репозиторий. В случае использования системы компьютерной верстки `LaTeX` также загрузить исходный файл `report.tex`.

Вариант задания:



Исходный код программы:

```
#include <windows.h>
#include <stdio.h>
#include <vector>
#include "lab3.h"

const int Q_MULTIPLIER = 4;

HANDLE sem_b = NULL;
HANDLE sem_c = NULL;
HANDLE sem_d = NULL;

struct ThreadParams {
    char thread_id;
    int total_iterations;
    int max_group_iterations;
    HANDLE self_sem;
    HANDLE next_sem;
};

DWORD WINAPI SimpleThreadFunc(LPVOID lpParam);
DWORD WINAPI AlternatingThreadFunc(LPVOID lpParam);

unsigned int lab3_thread_graph_id() {
    return 9;
}

const char* lab3_unsynchronized_threads() {
    return "fhi";
}

const char* lab3_sequential_threads() {
    return "bcd";
}

DWORD WINAPI SimpleThreadFunc(LPVOID lpParam) {
    ThreadParams* params = (ThreadParams*)lpParam;
    for (int i = 0; i < params->total_iterations; ++i) {
        printf("%c", params->thread_id);
        computation();
    }
    return 0;
}

DWORD WINAPI AlternatingThreadFunc(LPVOID lpParam) {
    ThreadParams* params = (ThreadParams*)lpParam;
    for (int i = 0; i < params->max_group_iterations; ++i) {
        WaitForSingleObject(params->self_sem, INFINITE);
        if (i < params->total_iterations) {
            printf("%c", params->thread_id);
            computation();
        }
    }
}
```

```

    }
    ReleaseSemaphore(params->next_sem, 1, NULL);
}
return 0;
}

int lab3_init() {
    std::vector<HANDLE> threads;

    const int iters_a = 1 * Q_MULTIPLIER;
    const int iters_b = 1 * Q_MULTIPLIER;
    const int iters_c = 2 * Q_MULTIPLIER;
    const int iters_d = 1 * Q_MULTIPLIER;
    const int iters_e = 1 * Q_MULTIPLIER;
    const int iters_f = 1 * Q_MULTIPLIER;
    const int iters_g = 1 * Q_MULTIPLIER;
    const int iters_h = 1 * Q_MULTIPLIER;
    const int iters_i = 1 * Q_MULTIPLIER;
    const int iters_k = 1 * Q_MULTIPLIER;

    int max_bcd_iters = max(iters_b, max(iters_c, iters_d));

    // Step 1: A
    ThreadParams paramsA = { 'a', iters_a, 0, NULL, NULL };
    DWORD tid;
    HANDLE hA = CreateThread(NULL, 0, SimpleThreadFunc, &paramsA, 0, &tid);
    if (!hA) return 1;
    threads.push_back(hA);
    WaitForSingleObject(hA, INFINITE);
    CloseHandle(hA);
    //printf("\n");

    // Step 2: B, C, D with semaphores
    sem_b = CreateSemaphore(NULL, 1, 1, NULL);
    sem_c = CreateSemaphore(NULL, 0, 1, NULL);
    sem_d = CreateSemaphore(NULL, 0, 1, NULL);
    if (!sem_b || !sem_c || !sem_d) return 2;

    ThreadParams paramsB = { 'b', iters_b, max_bcd_iters, sem_b, sem_c };
    ThreadParams paramsC = { 'c', iters_c, max_bcd_iters, sem_c, sem_d };
    ThreadParams paramsD = { 'd', iters_d, max_bcd_iters, sem_d, sem_b };

    HANDLE hB = CreateThread(NULL, 0, AlternatingThreadFunc, &paramsB, 0, &tid);
    HANDLE hC = CreateThread(NULL, 0, AlternatingThreadFunc, &paramsC, 0, &tid);
    HANDLE hD = CreateThread(NULL, 0, AlternatingThreadFunc, &paramsD, 0, &tid);
    if (!hB || !hC || !hD) return 3;

    threads.push_back(hB);
    threads.push_back(hC);
    threads.push_back(hD);

    HANDLE bcd_group[] = { hB, hC, hD };

```

```

WaitForMultipleObjects(3, bcd_group, TRUE, INFINITE);
//printf("\n");

// Step 3: E
ThreadParams paramsE = { 'e', iters_e, 0, NULL, NULL };
HANDLE hE = CreateThread(NULL, 0, SimpleThreadFunc, &paramsE, 0, &tid);
if (!hE) return 4;
threads.push_back(hE);
WaitForSingleObject(hE, INFINITE);
CloseHandle(hE);
//printf("\n");

// Step 4: F, H, I (unsynchronized)
ThreadParams paramsF = { 'f', iters_f, 0, NULL, NULL };
ThreadParams paramsH = { 'h', iters_h, 0, NULL, NULL };
ThreadParams paramsI = { 'i', iters_i, 0, NULL, NULL };

HANDLE hF = CreateThread(NULL, 0, SimpleThreadFunc, &paramsF, 0, &tid);
HANDLE hH = CreateThread(NULL, 0, SimpleThreadFunc, &paramsH, 0, &tid);
HANDLE hI = CreateThread(NULL, 0, SimpleThreadFunc, &paramsI, 0, &tid);

if (!hF || !hH || !hI) return 5;

threads.push_back(hF);
threads.push_back(hH);
threads.push_back(hI);

HANDLE group_fhi[] = { hF, hH, hI };
WaitForMultipleObjects(3, group_fhi, TRUE, INFINITE);
//printf("\n");

// Step 5: G
ThreadParams paramsG = { 'g', iters_g, 0, NULL, NULL };
HANDLE hG = CreateThread(NULL, 0, SimpleThreadFunc, &paramsG, 0, &tid);
if (!hG) return 6;
threads.push_back(hG);
WaitForSingleObject(hG, INFINITE);
CloseHandle(hG);
//printf("\n");

// Step 6: K
ThreadParams paramsK = { 'k', iters_k, 0, NULL, NULL };
HANDLE hK = CreateThread(NULL, 0, SimpleThreadFunc, &paramsK, 0, &tid);
if (!hK) return 7;
threads.push_back(hK);
WaitForSingleObject(hK, INFINITE);
CloseHandle(hK);
//printf("\n");

// Cleanup
for (HANDLE h : threads) {
    if (h) CloseHandle(h);
}

```

```
}  
  
if (sem_b) CloseHandle(sem_b);  
if (sem_c) CloseHandle(sem_c);  
if (sem_d) CloseHandle(sem_d);  
return 0;  
}
```



## Выводы

В ходе лабораторной работы были изучены базовые принципы создания многопоточных приложений в операционной системе Windows и освоены методы синхронизации потоков с использованием семафоров и функций ожидания.