МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №  43

ОТЧЁТ

ЗАЩИЩЁН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

### Старший преподаватель                                                      Поляк М.Д.

должность, уч. Степень, звание   подпись, дата                    инициалы, фамилия

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №3.

Синхронизация потоков средствами WinAPI.

по курсу: [Операционные системы](https://pro.guap.ru/inside/students/subjects/3154495)

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136                                                                                Бобрович Н. С.

                                                                         подпись, дата                      инициалы, фамилия

Санкт-Петербург 2024

1. **Цель работы:**

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

1. **Индивидуальное задание:**
2. 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:

выводить букву имени потока в консоль;

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.

1. 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 групу поток передает управление другому потоку после каждой итерации цикла (см. [задачу производителя и потребителя](https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem)). Таким образом потоки во второй группе выполняются в строгой очередности.
2. 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:

Функция unsigned int lab3\_thread\_graph\_id() должна возвращать номер графа запуска потоков, полученный из таблицы вариантов заданий.

Функция const char\* lab3\_unsynchronized\_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно без синхронизации.

Функция const char\* lab3\_sequential\_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно в строгой очередности друг за другом.

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.

Добавить любые другие необходимые для работы программы функции, переменные и подключаемые файлы.

Создавать функцию main() не нужно. В проекте уже имеется готовая функция main(), изменять ее нельзя. Она выполняет единственное действие: вызывает функцию lab3\_init().

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.

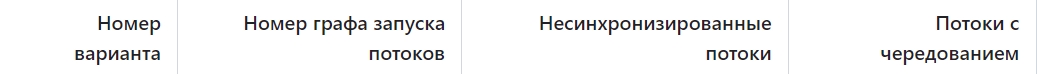
1. Подготовить отчет о выполнении лабораторной работы и загрузить его под именем report.pdf в репозиторий. В случае использования системы компьютерной верстки LaTeX также загрузить исходный файл report.tex.

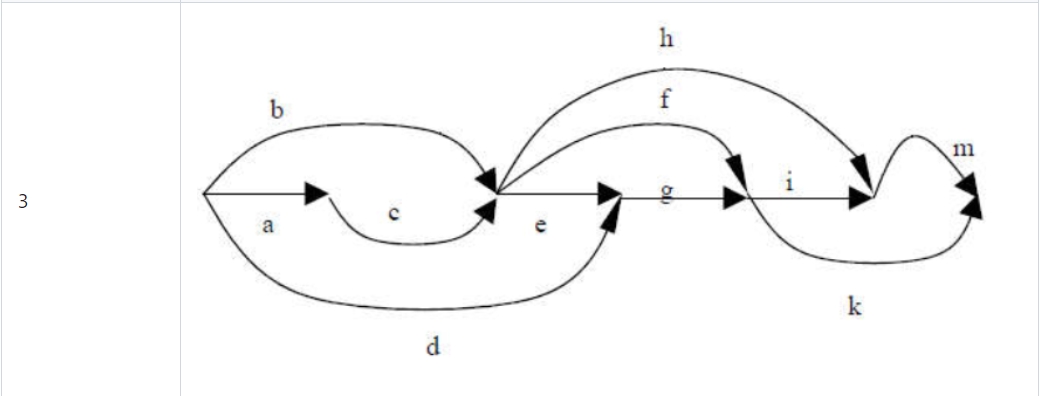
Последовательное выполнение потоков может обеспечиваться как за счет использования семафоров, так и с помощью функции WaitForSingleObject(). Запускать потоки можно все сразу в функции lab3\_init(), а можно и по одному (или группами) из других потоков.

В процессе своей работы каждый поток выводит свою букву в консоль. Оценка правильности выполнения лабораторной работы осуществляется следующим образом. Если потоки **a** и **b** согласно графу должны выполняться одновременно (параллельно), то в консоли должна присутствовать последовательность вида **abababab** (или схожая, например, **aabbba**); если потоки выполняются последовательно, то в консоли присутствует последовательность вида **aaaaabbbbbb**, причем после появления первой буквы **b**, буква **a** больше не должна появиться в консоли.

Количество букв, выводимых каждым потоком в консоль, должно быть пропорционально числу интервалов (длине дуги), соответствующей данному потоку на графе. При этом количество символов, выводимых в консоль каждым из потоков, должно быть не меньше чем 3Q и не больше чем 5Q, где Q - количество интервалов на графе, в течении которых выполняется поток. Множитель перед величиной Q следует выбрать одинаковым для всех потоков, задав его равным 3, 4 или 5.

1. **Граф запуска потоков:**

Скриншот 29-05-2024 194325



1. **Результат выполнения работы:**

bdabdadbabcdbcdbcddhefhehfddefhfghfgfhghikiikhhkkmmkmk

1. **Исходный код программы с комментариями:**

#include "lab3.h"

#include <windows.h>

#include <stdio.h>

// Эта функция возвращает ID графа потоков для текущей лабораторной работы

unsigned int lab3\_thread\_graph\_id()

{

return 3;

}

// Эта функция возвращает имена несинхронизированных потоков

const char\* lab3\_unsynchronized\_threads()

{

return "defh";

}

// Эта функция возвращает имена последовательных потоков

const char\* lab3\_sequential\_threads()

{

return "bcd";

}

// Глобальная переменная для хранения несинхронизированного блокиратора

HANDLE unsynchronizedLock;

// Функция для обработки производства и вычислений для несинхронизированных потоков

void productAndCompute(const std::string& charOut)

{

for (int i = 0; i < 3; ++i) {

auto res = WaitForSingleObject(unsynchronizedLock, INFINITE);

if (res == WAIT\_OBJECT\_0)

{

std::cout << charOut << std::flush;

ReleaseMutex(unsynchronizedLock);

}

computation();

}

}

// Функция для обработки последовательных потоков

void handleSequentialThreads(const std::string& charOut, HANDLE current, HANDLE next)

{

for (int i = 0; i < 3; ++i) {

auto res = WaitForSingleObject(current, INFINITE);

if (res == WAIT\_OBJECT\_0)

{

std::cout << charOut << std::flush;

ReleaseSemaphore(next, 1, NULL);

}

computation();

}

}

// Глобальные переменные для потоков задач

HANDLE taskBS;

void taskC();

void taskA() {

// Выполняем производство и вычисления для потока 'a'

productAndCompute("a");

// Освобождаем семафор taskBS, чтобы следующая задача могла начаться

ReleaseSemaphore(taskBS, 1, NULL);

// Создаем новый поток для выполнения taskC

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskC, NULL, 0, NULL);

// Выходим из текущего потока

return ExitThread(0);

}

HANDLE taskBSSync, taskCSSync;

void taskD();

void taskH();

void taskB()

{

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskA, NULL, 0, NULL);

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskD, NULL, 0, NULL);

productAndCompute("b");

WaitForSingleObject(taskBS, INFINITE);

WaitForSingleObject(taskBS, INFINITE);

handleSequentialThreads("b", taskBSSync, taskCSSync);

WaitForSingleObject(taskBS, INFINITE);

auto pthr = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskH, NULL, 0, NULL);

WaitForSingleObject(pthr, INFINITE);

CloseHandle(pthr);

return ExitThread(0);

}

HANDLE taskDSSync;

void taskC()

{

handleSequentialThreads("c", taskCSSync, taskDSSync);

return ExitThread(0);

}

HANDLE taskES, taskHS, taskFS;

void taskD()

{

productAndCompute("d");

ReleaseSemaphore(taskBS, 1, NULL);

handleSequentialThreads("d", taskDSSync, taskBSSync);

ReleaseSemaphore(taskBS, 1, NULL);

productAndCompute("d");

ReleaseSemaphore(taskES, 1, NULL);

ReleaseSemaphore(taskHS, 1, NULL);

ReleaseSemaphore(taskFS, 1, NULL);

return ExitThread(0);

}

void taskG();

void taskE()

{

productAndCompute("e");

ReleaseSemaphore(taskFS, 1, NULL);

ReleaseSemaphore(taskHS, 1, NULL);

WaitForSingleObject(taskES, INFINITE);

WaitForSingleObject(taskES, INFINITE);

WaitForSingleObject(taskES, INFINITE);

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskG, NULL, 0, NULL);

return ExitThread(0);

}

void taskI();

void taskF()

{

productAndCompute("f");

ReleaseSemaphore(taskES, 1, NULL);

ReleaseSemaphore(taskHS, 1, NULL);

WaitForSingleObject(taskFS, INFINITE);

WaitForSingleObject(taskFS, INFINITE);

WaitForSingleObject(taskFS, INFINITE);

productAndCompute("f");

ReleaseSemaphore(taskHS, 1, NULL);

WaitForSingleObject(taskFS, INFINITE);

WaitForSingleObject(taskFS, INFINITE);

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskI, NULL, 0, NULL);

return ExitThread(0);

}

void taskG()

{

// Выполняем производство и вычисления для потока 'g'

productAndCompute("g");

// Освобождаем семафоры taskHS и taskFS, чтобы следующие задачи могли начаться

ReleaseSemaphore(taskHS, 1, NULL);

ReleaseSemaphore(taskFS, 1, NULL);

// Выходим из текущего потока

return ExitThread(0);

}

HANDLE taskKS;

void taskM();

void taskH()

{

// Создаем новые потоки для выполнения taskF и taskE

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskF, NULL, 0, NULL);

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskE, NULL, 0, NULL);

// Выполняем производство и вычисления для потока 'h'

productAndCompute("h");

// Освобождаем семафоры taskFS и taskES, чтобы следующие задачи могли начаться

ReleaseSemaphore(taskFS, 1, NULL);

ReleaseSemaphore(taskES, 1, NULL);

// Ждем, пока семафор taskHS будет освобожден три раза

WaitForSingleObject(taskHS, INFINITE);

WaitForSingleObject(taskHS, INFINITE);

WaitForSingleObject(taskHS, INFINITE);

// Выполняем производство и вычисления для потока 'h' снова

productAndCompute("h");

// Освобождаем семафор taskFS, чтобы следующая задача могла начаться

ReleaseSemaphore(taskFS, 1, NULL);

// Ждем, пока семафор taskHS будет освобожден еще два раза

WaitForSingleObject(taskHS, INFINITE);

WaitForSingleObject(taskHS, INFINITE);

// Выполняем производство и вычисления для потока 'h' еще раз

productAndCompute("h");

// Освобождаем семафор taskKS, чтобы следующая задача могла начаться

ReleaseSemaphore(taskKS, 1, NULL);

// Ждем, пока семафор taskHS будет освобожден еще два раза

WaitForSingleObject(taskHS, INFINITE);

WaitForSingleObject(taskHS, INFINITE);

// Создаем новый поток для выполнения taskM

auto pthr = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskM, NULL, 0, NULL);

// Ждем, пока новый поток завершится

WaitForSingleObject(pthr, INFINITE);

// Закрываем дескриптор нового потока

CloseHandle(pthr);

// Выходим из текущего потока

return ExitThread(0);

}

void taskK();

void taskI()

{

// Создаем новый поток для выполнения taskK

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskK, NULL, 0, NULL);

// Выполняем производство и вычисления для потока 'i'

productAndCompute("i");

// Освобождаем семафоры taskKS и taskHS, чтобы следующие задачи могли начаться

ReleaseSemaphore(taskKS, 1, NULL);

ReleaseSemaphore(taskHS, 1, NULL);

// Выходим из текущего потока

return ExitThread(0);

}

HANDLE taskMS;

void taskK()

{

// Выполняем производство и вычисления для потока 'k'

productAndCompute("k");

// Освобождаем семафор taskHS, чтобы следующая задача могла начаться

ReleaseSemaphore(taskHS, 1, NULL);

// Ждем, пока семафор taskKS будет освобожден два раза

WaitForSingleObject(taskKS, INFINITE);

WaitForSingleObject(taskKS, INFINITE);

// Выполняем производство и вычисления для потока 'k' снова

productAndCompute("k");

// Освобождаем семафор taskMS, чтобы следующая задача могла начаться

ReleaseSemaphore(taskMS, 1, NULL);

// Выходим из текущего потока

return ExitThread(0);

}

void taskM()

{

// Выполняем производство и вычисления для потока 'm'

productAndCompute("m");

// Ждем, пока семафор taskMS будет освобожден

WaitForSingleObject(taskMS, INFINITE);

// Выходим из текущего потока

return ExitThread(0);

}

int lab3\_init()

{

// Создаем семафоры для синхронизации потоков

taskKS = CreateSemaphore(NULL, 0, 2, NULL);

taskBS = CreateSemaphore(NULL, 0, 2, NULL);

taskMS = CreateSemaphore(NULL, 0, 2, NULL);

taskES = CreateSemaphore(NULL, 0, 3, NULL);

taskHS = CreateSemaphore(NULL, 0, 3, NULL);

taskFS = CreateSemaphore(NULL, 0, 3, NULL);

taskBSSync = CreateSemaphore(NULL, 1, 1, NULL);

taskCSSync = CreateSemaphore(NULL, 0, 1, NULL);

taskDSSync = CreateSemaphore(NULL, 0, 1, NULL);

// Создаем несинхронизированный блокиратор

unsynchronizedLock = CreateMutex(NULL, FALSE, NULL);

// Создаем поток для выполнения задачи taskB

auto pthr = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)taskB, NULL, 0, NULL);

// Ждем, пока поток taskB завершится

WaitForSingleObject(pthr, INFINITE);

// Закрываем дескрипторы семафоров и блокиратора

CloseHandle(pthr);

CloseHandle(taskKS);

CloseHandle(taskBS);

CloseHandle(taskMS);

CloseHandle(taskES);

CloseHandle(taskHS);

CloseHandle(taskFS);

CloseHandle(taskBSSync);

CloseHandle(taskCSSync);

CloseHandle(taskDSSync);

// Возвращаем 0 для успешной инициализации

return 0;

}

1. **Выводы:**

Познакомился с многопоточным программированием и методами синхронизации потоков средствами Windows API.