МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЁТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Старший преподаватель | |  |  | | | |  | | М. Д. Поляк |
| должность, уч. степень, звание | |  | подпись, дата | | | |  | | инициалы, фамилия |
| ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №4 | | | | | | | | | | |
| Разработка многопоточног приложения в ОС Windows | | | | | | | | | | |
| по дисциплине: Операционные системы | | | | | | | | | | |
|  | | | | | | | | | | |
| РАБОТУ ВЫПОЛНИЛА | | | | | | | | | | |
| СТУДЕНТКА ГР. | 4931 | | |  | 18.04.2022 |  | | Е.Ю. Ильченко | | |
|  |  | | |  | подпись, дата |  | | инициалы, фамилия | | |
|  |  | | |  |  |  | |  | | |

Санкт-Петербург 2022

## Вариант 11

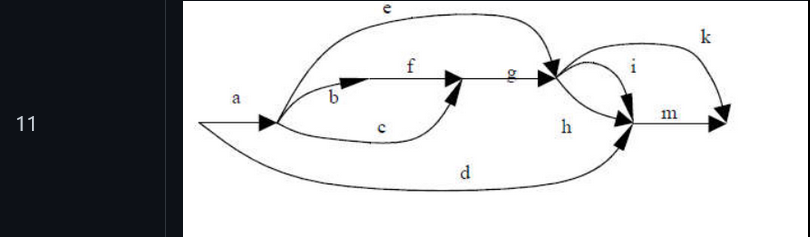
## Цель:

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

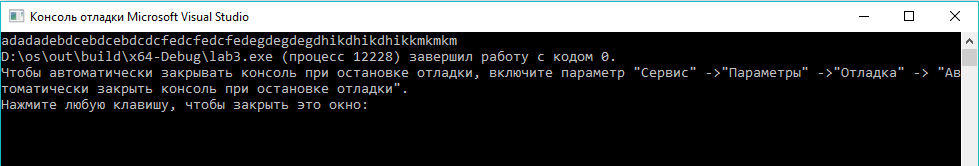
#### Задание:

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:
   1. выводить букву имени потока в консоль;
   2. 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 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.
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.). Для этого необходимо написать код в файле lab3.cpp:
   1. Функция unsigned int lab3\_thread\_graph\_id() должна возвращать номер графа запуска потоков, полученный из таблицы вариантов заданий.
   2. Функция const char\* lab3\_unsynchronized\_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно без синхронизации.
   3. Функция const char\* lab3\_sequential\_threads() должна возвращать строку, состоящую из букв потоков, выполняющихся параллельно в строгой очередности друг за другом.
   4. 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.
   5. Добавить любые другие необходимые для работы программы функции, переменные и подключаемые файлы.
   6. Создавать функцию main() не нужно. В проекте уже имеется готовая функция main(), изменять ее нельзя. Она выполняет единственное действие: вызывает функцию lab3\_init().
   7. 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер варианта | Номер графа запуска потоков | Интервалы с несинхронизированными потоками | Интервалы с чередованием потоков |
| 11 | 11 | |  |  | | --- | --- | | bcde |  | | deg |



**Результат выполнения работы программы:**

****

**Исходный код программы:**

#include "lab3.h"

#include"windows.h"

//

// lab3 code should be located here!

//

#define THREAD\_COUNT 11

DWORD ThreadID;

HANDLE Thread[THREAD\_COUNT];

HANDLE mutex;

HANDLE semD, semE, semF, semC, semG, semH, semI, semK, semM;

DWORD WINAPI Thread\_a(LPVOID);

DWORD WINAPI Thread\_d(LPVOID);

DWORD WINAPI Thread\_e(LPVOID);

DWORD WINAPI Thread\_b(LPVOID);

DWORD WINAPI Thread\_c(LPVOID);

DWORD WINAPI Thread\_f(LPVOID);

DWORD WINAPI Thread\_g(LPVOID);

DWORD WINAPI Thread\_k(LPVOID);

DWORD WINAPI Thread\_i(LPVOID);

DWORD WINAPI Thread\_h(LPVOID);

DWORD WINAPI Thread\_m(LPVOID);

unsigned int lab3\_thread\_graph\_id()

{

return 11;

}

const char\* lab3\_unsynchronized\_threads()

{

return "bcde";

}

const char\* lab3\_sequential\_threads()

{

return "deg";

}

DWORD WINAPI Thread\_a(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//1-й интервал

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

{

WaitForSingleObject(mutex, INFINITE);

std::cout << "a" << std::flush;

computation();

ReleaseMutex(mutex);

}

return 0;

}

DWORD WINAPI Thread\_d(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//1-й интервал

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

{

WaitForSingleObject(mutex, INFINITE);

std::cout << "d" << std::flush;

computation();

ReleaseMutex(mutex);

}

// ожидание завершения потока A

WaitForSingleObject(Thread[0], INFINITE);

//создание потока e

Thread[2] = CreateThread(NULL, 0, Thread\_e, NULL, 0, &ThreadID);

if (Thread[2] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//создание потока b

Thread[3] = CreateThread(NULL, 0, Thread\_b, NULL, 0, &ThreadID);

if (Thread[3] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//создание потока c

Thread[4] = CreateThread(NULL, 0, Thread\_c, NULL, 0, &ThreadID);

if (Thread[4] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//2-й интервал

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

{

WaitForSingleObject(mutex, INFINITE);

std::cout << "d" << std::flush;

computation();

ReleaseMutex(mutex);

}

// ожидание завершения потока B

WaitForSingleObject(Thread[3], INFINITE);

//создание потока F

Thread[5] = CreateThread(NULL, 0, Thread\_f, NULL, 0, &ThreadID);

if (Thread[5] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//3-й интервал

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

{

WaitForSingleObject(semD, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "d" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semC, 1, NULL);

}

// ожидание завершения потока F

WaitForSingleObject(Thread[5], INFINITE);

// ожидание завершения потока C

WaitForSingleObject(Thread[4], INFINITE);

//создание потока G

Thread[6] = CreateThread(NULL, 0, Thread\_g, NULL, 0, &ThreadID);

if (Thread[5] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//4-й интервал

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

{

WaitForSingleObject(semD, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "d" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semE, 1, NULL);

}

// ожидание завершения потока E

WaitForSingleObject(Thread[2], INFINITE);

// ожидание завершения потока G

WaitForSingleObject(Thread[6], INFINITE);

//создание потока K

Thread[7] = CreateThread(NULL, 0, Thread\_k, NULL, 0, &ThreadID);

if (Thread[7] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//создание потока I

Thread[8] = CreateThread(NULL, 0, Thread\_i, NULL, 0, &ThreadID);

if (Thread[8] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//создание потока H

Thread[9] = CreateThread(NULL, 0, Thread\_h, NULL, 0, &ThreadID);

if (Thread[9] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//5-й интервал

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

{

WaitForSingleObject(semD, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "d" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semH, 1, NULL);

}

// ожидание завершения потока H

WaitForSingleObject(Thread[9], INFINITE);

// ожидание завершения потока I

WaitForSingleObject(Thread[8], INFINITE);

// ожидание завершения потока K

WaitForSingleObject(Thread[7], INFINITE);

return 0;

}

DWORD WINAPI Thread\_e(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//2-й интервал

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

{

WaitForSingleObject(mutex, INFINITE);

std::cout << "e" << std::flush;

computation();

ReleaseMutex(mutex);

}

//3-й интервал

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

{

WaitForSingleObject(semE, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "e" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semD, 1, NULL);

}

//4-й интервал

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

{

WaitForSingleObject(semE, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "e" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semG, 1, NULL);

}

return 0;

}

DWORD WINAPI Thread\_b(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//2-й интервал

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

{

WaitForSingleObject(mutex, INFINITE);

std::cout << "b" << std::flush;

computation();

ReleaseMutex(mutex);

}

return 0;

}

DWORD WINAPI Thread\_c(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//2-й интервал

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

{

WaitForSingleObject(mutex, INFINITE);

std::cout << "c" << std::flush;

computation();

ReleaseMutex(mutex);

}

//3-й интервал

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

{

WaitForSingleObject(semC, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "c" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semF, 1, NULL);

}

return 0;

}

DWORD WINAPI Thread\_f(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//3-й интервал

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

{

WaitForSingleObject(semF, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "f" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semE, 1, NULL);

}

return 0;

}

DWORD WINAPI Thread\_g(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//4-й интервал

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

{

WaitForSingleObject(semG, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "g" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semD, 1, NULL);

}

return 0;

}

DWORD WINAPI Thread\_h(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//5-й интервал

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

{

WaitForSingleObject(semH, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "h" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semI, 1, NULL);

}

return 0;

}

DWORD WINAPI Thread\_i(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//5-й интервал

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

{

WaitForSingleObject(semI, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "i" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semK, 1, NULL);

}

return 0;

}

DWORD WINAPI Thread\_k(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//5-й интервал

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

{

WaitForSingleObject(semK, INFINITE);

WaitForSingleObject(mutex, INFINITE);

std::cout << "k" << std::flush;

computation();

ReleaseMutex(mutex);

ReleaseSemaphore(semD, 1, NULL);

}

//создание потока M

Thread[10]= CreateThread(NULL, 0, Thread\_m, NULL, 0, &ThreadID);

if (Thread[10]== NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//6-й интервал

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

{

WaitForSingleObject(mutex, INFINITE);

std::cout << "k" << std::flush;

computation();

ReleaseMutex(mutex);

}

// ожидание завершения потока M

WaitForSingleObject(Thread[10], INFINITE);

return 0;

}

DWORD WINAPI Thread\_m(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

//6-й интервал

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

{

WaitForSingleObject(mutex, INFINITE);

std::cout << "m" << std::flush;

computation();

ReleaseMutex(mutex);

}

return 0;

}

int lab3\_init()

{

mutex = CreateMutex(NULL, 0, NULL);

if (mutex == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

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

if (semD == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

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

if (semE == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

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

if (semF == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

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

if (semC == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

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

if (semG == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

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

if (semH == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

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

if (semI == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

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

if (semK == NULL)

{

std::cout << "CreateSemaphore error: " << GetLastError();

return 1;

}

//создание потока a

Thread[0] = CreateThread(NULL, 0, Thread\_a, NULL, 0, &ThreadID);

if (Thread[0] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//создание потока d

Thread[1] = CreateThread(NULL, 0, Thread\_d, NULL, 0, &ThreadID);

if (Thread[1] == NULL)

{

std::cout << "CreateThread error: " << GetLastError();

return 1;

}

//ожидание завершения потока d

WaitForSingleObject(Thread[1], INFINITE);

CloseHandle(mutex);

CloseHandle(semD);

CloseHandle(semE);

CloseHandle(semF);

CloseHandle(semC);

CloseHandle(semG);

CloseHandle(semH);

CloseHandle(semI);

CloseHandle(semK);

return 0;

}

**Выводы:** В процессе выполнения лабораторной работы ознакомились с многопоточным программированием и методами синхронизации потоков средствами Windows API.