КАФЕДРА №

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

РАЗРАБОТКА МНОГОПОТОЧНОГО ПРИЛОЖЕНИЯ В ОС WINDOWS.

по курсу: Операционные системы

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № _____ _____ _____
подпись, дата инициалы, фамилия

Санкт-Петербург 2024

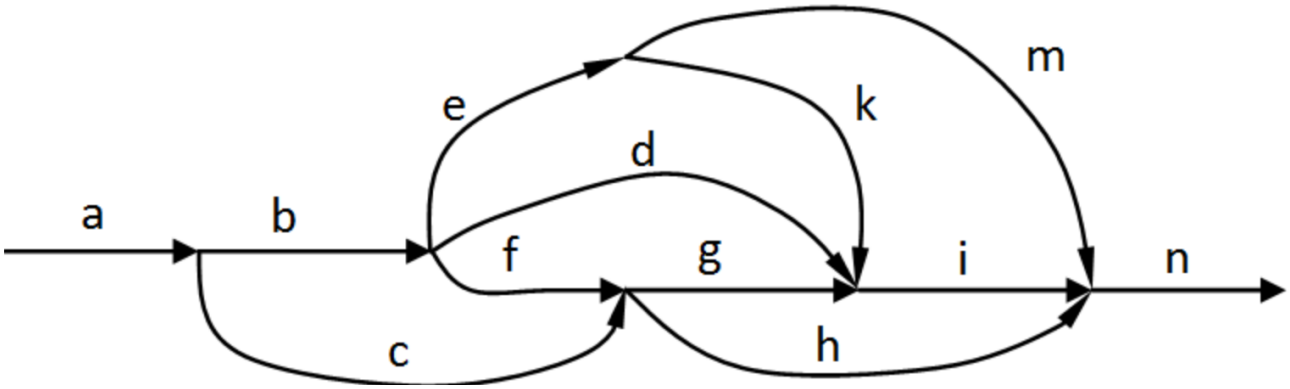
Цель работы

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

Задание на лабораторную работу

Номер варианта	Номер графа запуска потоков	Интервалы с несинхронизированными потоками	Интервалы с чередованием потоков
13	13	cdef	him

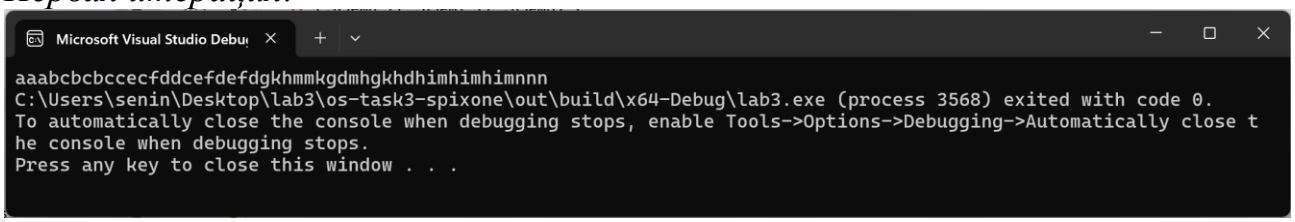
Граф запуска потоков



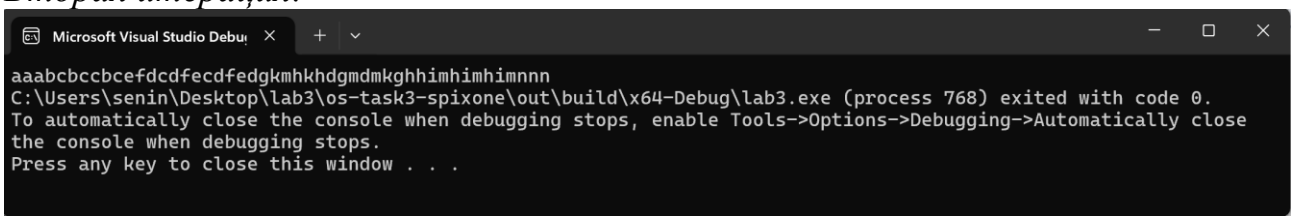
Результат выполнения работы

В результате выполнения программы выводятся следующие последовательности символов, которые полностью соответствуют заданию по варианту 13.

Первая итерация:



Вторая итерация:



Исходный код программы с комментариями

```
#include <windows.h>
#include <iostream>
#include "lab3.h" // Подключение файла с функцией computation()

using namespace std;

unsigned int lab3_thread_graph_id()
{
   return 13;
}

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

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

const int numThreads = 12;
HANDLE hThreads[numThreads];
DWORD ThreadId;

HANDLE hMutex;
HANDLE hSemH, hSemI, hSemM, hSemEnd;

// Отдельные функции для каждого потока
DWORD WINAPI ThreadA(LPVOID); // 0
DWORD WINAPI ThreadB(LPVOID); // 1
DWORD WINAPI ThreadC(LPVOID); // 2
DWORD WINAPI ThreadD(LPVOID); // 3
DWORD WINAPI ThreadE(LPVOID); // 4
DWORD WINAPI ThreadF(LPVOID); // 5
DWORD WINAPI ThreadG(LPVOID); // 6
DWORD WINAPI ThreadH(LPVOID); // 7
DWORD WINAPI ThreadI(LPVOID); // 8
DWORD WINAPI ThreadK(LPVOID); // 9
DWORD WINAPI ThreadM(LPVOID); // 10
DWORD WINAPI ThreadN(LPVOID); // 11

// Функции для потоков
DWORD WINAPI ThreadA(LPVOID lpParam) {
   for (int i = 0; i < 3; i++) {
      WaitForSingleObject(hMutex, INFINITE); // Блокировка мьютекса
      cout << 'a' << flush;
      ReleaseMutex(hMutex); // Разблокировка мьютекса
      computation();
   }
```

```
    hThreads[1] = CreateThread(NULL, 0, ThreadB, NULL, 0, &ThreadId);
    hThreads[2] = CreateThread(NULL, 0, ThreadC, hThreads[1], 0, &ThreadId);

    return 0;
}

DWORD WINAPI ThreadB(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'b' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    return 0;
}

DWORD WINAPI ThreadC(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'c' << flush;
        ReleaseMutex(hMutex);
        computation();
    }


    HANDLE hThreadB = (HANDLE)lpParam;  // Принимаем дескриптор потока B как параметр
    WaitForSingleObject(hThreadB, INFINITE);  // Ожидание завершения потока B
    // Закрываем дескриптор потока B
    CloseHandle(hThreads[1]);

    // Запуск потоков D, E, F
    hThreads[4] = CreateThread(NULL, 0, ThreadE, hThreads[2], 0, &ThreadId); // в E передается
C
    hThreads[5] = CreateThread(NULL, 0, ThreadF, hThreads[4], 0, &ThreadId); // в F передается
E
    hThreads[3] = CreateThread(NULL, 0, ThreadD, hThreads[5], 0, &ThreadId); // в D передается
F

    // Второй цикл потока C
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'c' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    return 0;
}

DWORD WINAPI ThreadD(LPVOID lpParam) {
```

```
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'd' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    HANDLE hThreadF = (HANDLE)lpParam;  // Принимаем дескриптор потока C как параметр
    WaitForSingleObject(hThreadF, INFINITE);  // Ожидание завершения потока C
    CloseHandle(hThreads[5]);

    // Запуск потоков G, H, K, M
    hThreads[6] = CreateThread(NULL, 0, ThreadG, hThreads[3], 0, &ThreadId);
    hThreads[9] = CreateThread(NULL, 0, ThreadK, hThreads[6], 0, &ThreadId);
    hThreads[7] = CreateThread(NULL, 0, ThreadH, hThreads[9], 0, &ThreadId);
    hThreads[10] = CreateThread(NULL, 0, ThreadM, NULL, 0, &ThreadId);

    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'd' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    return 0;
}

DWORD WINAPI ThreadE(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'e' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    HANDLE hThreadC = (HANDLE)lpParam;  // Принимаем дескриптор потока C как параметр
    WaitForSingleObject(hThreadC, INFINITE);  // Ожидание завершения потока C
    CloseHandle(hThreads[2]);

    return 0;
}

DWORD WINAPI ThreadF(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'f' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    HANDLE hThreadE = (HANDLE)lpParam;  // Принимаем дескриптор потока E как параметр
    WaitForSingleObject(hThreadE, INFINITE);  // Ожидание завершения потока E
```

```
    CloseHandle(hThreads[4]);

    return 0;
}

DWORD WINAPI ThreadG(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'g' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    HANDLE hThreadD = (HANDLE)lpParam;  // Принимаем дескриптор потока D как
параметр
    WaitForSingleObject(hThreadD, INFINITE);  // Ожидание завершения потока D
    CloseHandle(hThreads[3]);

    return 0;
}

DWORD WINAPI ThreadH(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'h' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    HANDLE hThreadK = (HANDLE)lpParam;  // Принимаем дескриптор потока K как
параметр
    WaitForSingleObject(hThreadK, INFINITE);  // Ожидание завершения потока K
    CloseHandle(hThreads[9]);

    hThreads[8] = CreateThread(NULL, 0, ThreadI, NULL, 0, &ThreadId);

    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hSemH, INFINITE); // Ждать разблокировки
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'h' << flush;
        ReleaseMutex(hMutex);
        computation();
        ReleaseSemaphore(hSemI, 1, NULL); // Разблокировать поток I
    }

    return 0;
}

DWORD WINAPI ThreadI(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hSemI, INFINITE); // Ждать разблокировки
        WaitForSingleObject(hMutex, INFINITE);
```

```
        cout << 'i' << flush;
        ReleaseMutex(hMutex);
        computation();
        ReleaseSemaphore(hSemM, 1, NULL); // Разблокировать поток M
    }

    return 0;
}

DWORD WINAPI ThreadK(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'k' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    HANDLE hThreadG = (HANDLE)lpParam;  // Принимаем дескриптор потока G как
параметр
    WaitForSingleObject(hThreadG, INFINITE);  // Ожидание завершения потока G
    CloseHandle(hThreads[6]);

    return 0;
}

DWORD WINAPI ThreadM(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'm' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    ReleaseSemaphore(hSemH, 1, NULL); // Разблокировать поток H

    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hSemM, INFINITE); // Ждать разблокировки
        WaitForSingleObject(hMutex, INFINITE);
        cout << 'm' << flush;
        ReleaseMutex(hMutex);
        computation();
        ReleaseSemaphore(hSemH, 1, NULL); // Разблокировать поток H
    }

    hThreads[11] = CreateThread(NULL, 0, ThreadN, NULL, 0, &ThreadId);

    return 0;
}

DWORD WINAPI ThreadN(LPVOID lpParam) {
    for (int i = 0; i < 3; i++) {
        WaitForSingleObject(hMutex, INFINITE);
```

```
        cout << 'n' << flush;
        ReleaseMutex(hMutex);
        computation();
    }

    ReleaseSemaphore(hSemEnd, 1, NULL);

    return 0;
}

int lab3_init() {
    // Создание мьютекса
    hMutex = CreateMutex(NULL, FALSE, NULL);
    if (!hMutex) {
        cout << "Error creating mutex." << endl;
        return 1; // Возвращаем код ошибки
    }

    // Создание семафоров
    hSemH = CreateSemaphore(NULL, 0, 1, NULL);
    hSemI = CreateSemaphore(NULL, 0, 1, NULL);
    hSemM = CreateSemaphore(NULL, 0, 1, NULL);

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

    // Проверка создания семафоров
    if (!hSemH || !hSemI || !hSemM) {
        cout << "Error creating semaphores." << endl;
        return 1; // Возвращаем код ошибки
    }

    // Запускаем поток A изначально
    hThreads[0] = CreateThread(NULL, 0, ThreadA, NULL, 0, &ThreadId);

    // Ожидание завершения всех потоков (потока N)
    WaitForSingleObject(hSemEnd, INFINITE);

    CloseHandle(hThreads[7]);
    CloseHandle(hThreads[8]);
    CloseHandle(hThreads[10]);
    CloseHandle(hThreads[11]);

    CloseHandle(hMutex); // Закрываем дескриптор мьютекса

    // Закрываем дескрипторы семафоров после завершения всех потоков
    CloseHandle(hSemH);
    CloseHandle(hSemI);
    CloseHandle(hSemM);

    return 0;
}
```

Вывод

В процессе работы я получил практический опыт использования многопоточного программирования и освоил методы синхронизации потоков, такие как мьютексы и семафоры, с помощью Windows API. Эти навыки позволили мне глубже понять принципы управления потоками и их взаимодействие для эффективной работы приложений.