МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский государственный университет
аэрокосмического приборостроения»

КАФЕДРА 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ

Ст. преподаватель		М.Д. Поляк
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. №	4134К		Самарин Д.В
		подпись, дата	инициалы, фамилия

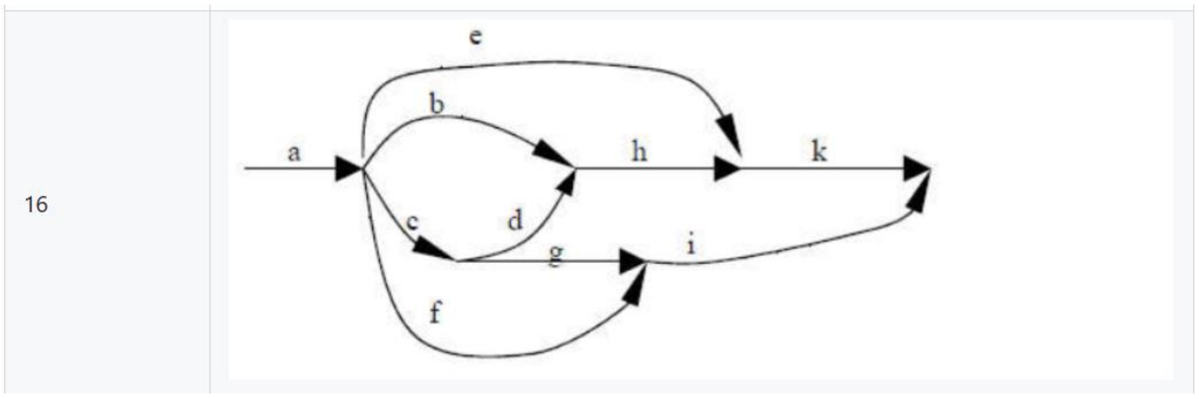
Санкт-Петербург 2024

Цель работы:

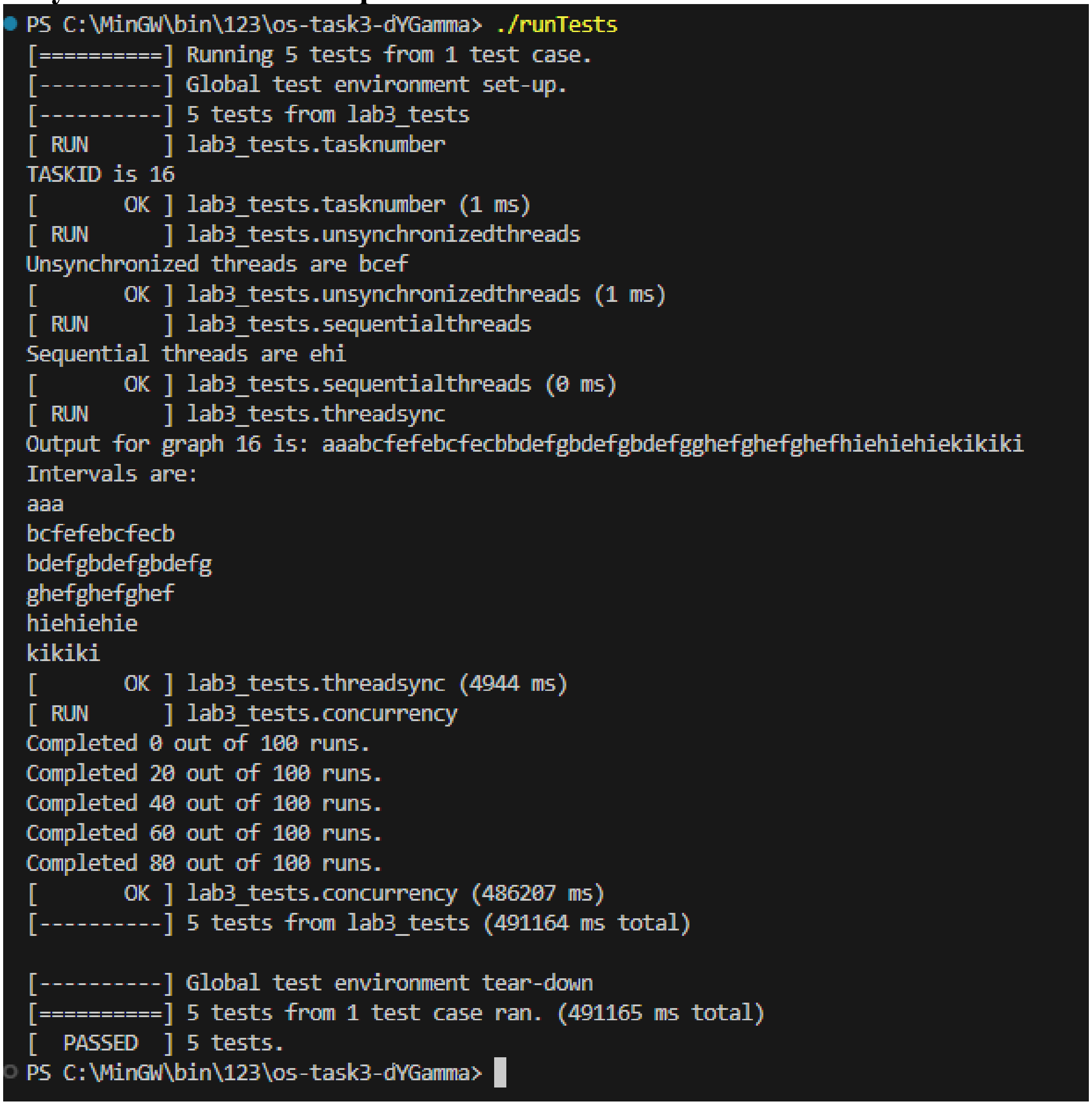
Знакомство с многопоточным программированием и методами синхронизации потоков средствами POSIX.

Индивидуальное задание:

Номер варианта	**Номер графа**	**Интервалы (несинх.)**	**Интервалы с чередованием потоков**
16	**16**	**bcef**	**ehi**



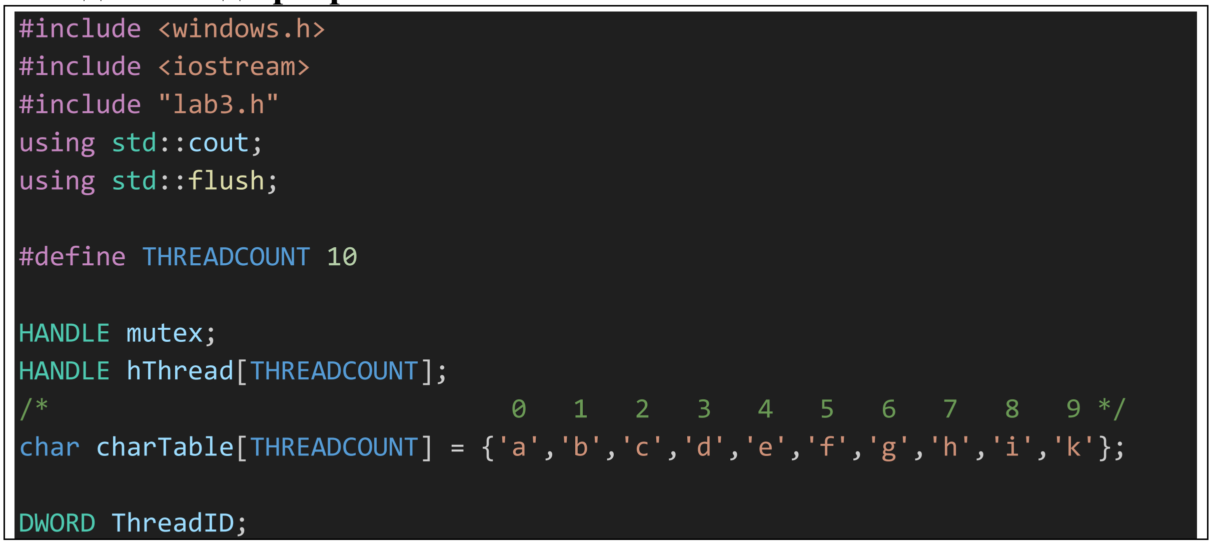
Результат выполнения работы:



```
PS C:\MinGW\bin\123\os-task3-dYGamma> ./runTests
[==========] Running 5 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 5 tests from lab3_tests
[ RUN      ] lab3_tests.tasknumber
TASKID is 16
[       OK ] lab3_tests.tasknumber (1 ms)
[ RUN      ] lab3_tests.unsynchronizedthreads
Unsynchronized threads are bcef
[       OK ] lab3_tests.unsynchronizedthreads (1 ms)
[ RUN      ] lab3_tests.sequentialthreads
Sequential threads are ehi
[       OK ] lab3_tests.sequentialthreads (0 ms)
[ RUN      ] lab3_tests.threadsync
Output for graph 16 is: aaabcfefebcfecbbdefgbdefgbdefgghefghefghefhiehiehiekikiki
Intervals are:
aaa
bcfefebcfecb
bdefgbdefgbdefg
ghefghefghef
hiehiehie
kikiki
[       OK ] lab3_tests.threadsync (4944 ms)
[ RUN      ] lab3_tests.concurrency
Completed 0 out of 100 runs.
Completed 20 out of 100 runs.
Completed 40 out of 100 runs.
Completed 60 out of 100 runs.
Completed 80 out of 100 runs.
[       OK ] lab3_tests.concurrency (486207 ms)
[----------] 5 tests from lab3_tests (491164 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test case ran. (491165 ms total)
[  PASSED  ] 5 tests.
PS C:\MinGW\bin\123\os-task3-dYGamma>
```

Исходный код программы:



```
#include <windows.h>
#include <iostream>
#include "lab3.h"
using std::cout;
using std::flush;

#define THREADCOUNT 10

HANDLE mutex;
HANDLE hThread[THREADCOUNT];
/*                              0   1   2   3   4   5   6   7   8   9 */
char charTable[THREADCOUNT] = {'a','b','c','d','e','f','g','h','i','k'};

DWORD ThreadID;
```

```
HANDLE semaphoreTable[10];
/*                               0  1  2  3  4  5  6  7  8  9 */
int semaphoreTableCount[10] = {0, 3, 0, 3, 9, 6, 6, 6, 6, 3};


unsigned int lab3_thread_graph_id()
{
    return 16;
}

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

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

DWORD WINAPI thread_b(LPVOID lpParam);
DWORD WINAPI thread_c(LPVOID lpParam);
DWORD WINAPI thread_d(LPVOID lpParam);
DWORD WINAPI thread_e(LPVOID lpParam);
DWORD WINAPI thread_f(LPVOID lpParam);
DWORD WINAPI thread_g(LPVOID lpParam);
DWORD WINAPI thread_h(LPVOID lpParam);
DWORD WINAPI thread_i(LPVOID lpParam);
DWORD WINAPI thread_k(LPVOID lpParam);

void thread_unsynchronized(int thread);
void thread_sequential(int threads);

DWORD WINAPI thread_a(LPVOID lpParam)
{
    for(int i = 0; i < 3; i++) {
        WaitForSingleObject(mutex, INFINITE);
        cout << charTable[(int)lpParam] << flush;
        ReleaseMutex(mutex);
        computation();
    }

    hThread[1] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_b, NULL,  0, &ThreadID);
    if( hThread[1] == NULL )
        cout << "CreateThread error: " << GetLastError();

    hThread[2] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_c, NULL,  0, &ThreadID);
```

```
    if( hThread[2] == NULL )
        cout << "CreateThread error: " << GetLastError();

    hThread[4] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_e, NULL,  0, &ThreadID);
    if( hThread[4] == NULL )
        cout << "CreateThread error: " << GetLastError();

    hThread[5] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_f, NULL,  0, &ThreadID);
    if( hThread[5] == NULL )
        cout << "CreateThread error: " << GetLastError();

    WaitForSingleObject(hThread[1], INFINITE);
    WaitForSingleObject(hThread[4], INFINITE);

    ExitThread(0);
}

DWORD WINAPI thread_b(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    thread_unsynchronized(1);

    WaitForSingleObject(hThread[2], INFINITE);
    WaitForSingleObject(hThread[4], 1000L);
    WaitForSingleObject(hThread[5], 1000L);

    hThread[3] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_d, NULL,  0, &ThreadID);
    if( hThread[3] == NULL )
        cout << "CreateThread error: " << GetLastError();
    hThread[6] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_g, NULL,  0, &ThreadID);
    if( hThread[6] == NULL )
        cout << "CreateThread error: " << GetLastError();

    ReleaseSemaphore(semaphoreTable[1], 1,NULL);
    thread_sequential(13);

    ExitThread(0);
}

DWORD WINAPI thread_c(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    thread_unsynchronized(2);
    ExitThread(0);
}

DWORD WINAPI thread_d(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    thread_sequential(34);

    ExitThread(0);
}

DWORD WINAPI thread_e(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    thread_unsynchronized(4);

    WaitForSingleObject(hThread[2], INFINITE);
    thread_sequential(45);

    hThread[7] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_h, NULL,  0, &ThreadID);
    if( hThread[7] == NULL )
        cout << "CreateThread error: " << GetLastError();

    ReleaseSemaphore(semaphoreTable[6], 1,NULL);
    thread_sequential(45);

    hThread[8] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_i, NULL,  0, &ThreadID);
    if( hThread[8] == NULL )
        cout << "CreateThread error: " << GetLastError();

    WaitForSingleObject(hThread[5], INFINITE);
    ReleaseSemaphore(semaphoreTable[7], 1,NULL);
    thread_sequential(47);

    WaitForSingleObject(hThread[7], INFINITE);
    WaitForSingleObject(hThread[8], INFINITE);
    ExitThread(0);
}

DWORD WINAPI thread_f(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    thread_unsynchronized(5);

    WaitForSingleObject(hThread[2], INFINITE);
    thread_sequential(56);

    thread_sequential(56);

    ExitThread(0);
}

DWORD WINAPI thread_g(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    WaitForSingleObject(hThread[2], INFINITE);
    thread_sequential(61);

    thread_sequential(67);

    ExitThread(0);
}

DWORD WINAPI thread_h(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    thread_sequential(74);

    thread_sequential(78);
    ExitThread(0);
}

DWORD WINAPI thread_i(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);

    thread_sequential(84);

    hThread[9] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_k, NULL,  0, &ThreadID);
    if( hThread[9] == NULL )
        cout << "CreateThread error: " << GetLastError();

    ReleaseSemaphore(semaphoreTable[9], 1,NULL);
    thread_sequential(89);

    WaitForSingleObject(hThread[9], INFINITE);
    ExitThread(0);
}

DWORD WINAPI thread_k(LPVOID lpParam)
{
    UNREFERENCED_PARAMETER(lpParam);
    thread_sequential(98);

    ExitThread(0);
}

void thread_unsynchronized(int thread)
{
    for(int i = 0; i < 3; i++) {
        WaitForSingleObject(mutex, INFINITE);
        cout << charTable[thread] << flush;
        ReleaseMutex(mutex);
        computation();
    }
}

void thread_sequential(int threads)
{
    DWORD dwWaitResult;
    BOOL bContinue;
    for(int i = 0; i < 3; i++)
    {
        bContinue=TRUE;
        while(bContinue)
        {
            dwWaitResult =
WaitForSingleObject(semaphoreTable[(int)threads/10],60000L);
            switch (dwWaitResult)
            {
                case WAIT_OBJECT_0:
                    WaitForSingleObject(mutex, INFINITE);
                    cout << charTable[(int)threads/10] << flush;
                    ReleaseMutex(mutex);
                    computation();

                    bContinue=FALSE;
                if(!ReleaseSemaphore(semaphoreTable[(int)threads%10],
1,NULL))
                {
                    cout << "ReleaseSemaphore error: " << GetLastError();
                }
                break;
                case WAIT_TIMEOUT:
                    cout << "Thread %d: wait timed out\n" <<
GetCurrentThreadId();
                break;
                }
        }
    }
}
```

```
int lab3_init()
{
    mutex = CreateMutex(NULL, FALSE, NULL);

    if(mutex == NULL)
    {
        cout << "CreateMutex error: " << GetLastError();
        return 1;
    }

    for(int i = 0; i < 10; i++)
    {
        if(i != 0 && i != 2)
        {
            semaphoreTable[i] = CreateSemaphore(NULL, 0,
semaphoreTableCount[i], NULL);
            if (semaphoreTable[i] == NULL)
            {
                cout << "CreateSemaphore " << i << " error : "
<<  GetLastError();
                return 1;
            }
        }
    }

    //a
    hThread[0] = CreateThread(NULL,0, (LPTHREAD_START_ROUTINE)
thread_a, 0,  0, &ThreadID);
    if( hThread[0] == NULL )
        cout << "CreateThread error: " << GetLastError();

    WaitForSingleObject(hThread[0], INFINITE);

    for(int i = 0; i < THREADCOUNT; i++)
        CloseHandle(hThread[i]);

    //close semaphore
    for(int i = 0; i < 10; i++)
    {
        if(i != 0 && i != 2)
            CloseHandle(semaphoreTable[i]);
    }

    CloseHandle(mutex);

    return 0;
}
```

Выводы

В ходе работы я познакомился с многопоточным программированием и методами синхронизации потоков средствами POSIX.