МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший 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. | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

2019

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

**Задание**

1. Вычислить номер варианта задания как остаток от деления **суммы** порядкового номера студента по списку в журнале и **числа 5** на количество вариантов заданий. Если остаток равен нулю, выбрать последнее задание.  
   **Nвар = (Nв\_списке) mod K**,  
   где Nвар - искомый номер варианта, Nв\_списке - порядковый номер студента по списку в журнале, K - количество вариантов заданий на данную лабораторную работу.
2. Выбрать граф запуска потоков в соответствии с вариантом задания. Вершины графа являются точками запуска/завершения потоков, дугами обозначены сами потоки, длину дуги следует интерпретировать как ориентировочное время выполнения потока.
3. Реализовать последовательно-параллельный запуск потоков в ОС Linux или Mac OS X с использованием средств POSIX для запуска и синхронизации потоков. 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 lab2.cpp:
   1. Функция unsigned int lab2\_task\_number() должна возвращать номер варианта, полученный на шаге 1.
   2. Функция int lab2\_init() заменяет собой функцию main(). В ней необходимо реализовать запуск потоков, инициализацию вспомогательных переменных (мьютексов, семафоров и т.п.). Перед выходом из функции lab2\_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.
   3. Добавить любые другие необходимые для работы программы функции, переменные и подключаемые файлы.
   4. Писать функцию main() не нужно. В проекте уже имеется готовая функция main(), изменять ее нельзя. Она выполняет единственное действие: вызывает функцию lab2\_init().
   5. Не следует изменять какие-либо файлы, кроме lab2.cpp. Также не следует создавать новые файлы и писать в них код, поскольку код из этих файлов не будет использоваться во время тестирования.
4. Самостоятельно выделить на графе две группы с выполняющимися параллельно потоками. Первая группа не синхронизирована, параллельное выполнение входящих в группу потоков происходит за счет использования искусственной задержки (см. [примеры](https://github.com/suai-os-2019/os-task2-Grishy/blob/master/examples/README.md) [1](https://github.com/suai-os-2019/os-task2-Grishy/blob/master/examples/README.md#%D0%9E%D1%80%D0%B3%D0%B0%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F-%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE-%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F-%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%B2-%D0%B1%D0%B5%D0%B7-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D1%81%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B2-%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8) и [2](https://github.com/suai-os-2019/os-task2-Grishy/blob/master/examples/README.md#%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D0%BC%D1%8C%D1%8E%D1%82%D0%B5%D0%BA%D1%81%D0%B0-%D0%B4%D0%BB%D1%8F-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B-%D1%81-%D0%BE%D0%B1%D1%89%D0%B8%D0%BC-%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D0%BE%D0%BC)). Величина задержки не должна быть как можно меньше. При выполнении операций ввода-вывода перед входом в критическую секцию потоки должны захватывать мьютекс. Вторая группа синхронизирована семафорами: входящий в групу поток передает управление другому потоку после каждой итерации (см. [пример 3](https://github.com/suai-os-2019/os-task2-Grishy/blob/master/examples/README.md#%D0%98%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%81%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80%D0%BE%D0%B2-%D0%B4%D0%BB%D1%8F-%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8-%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%B2)).
5. Последовательное выполнение потоков должно обеспечиваться за счет использования семафоров. Все потоки должны запускаться одновременно из функции lab2\_init(), без задержек, один за другим. Использовать функцию pthread\_join() разрешается только в функции lab2\_init() для ожидания завершения работы всех запущенных потоков.

|  |  |
| --- | --- |
| 9 | ÐÑÐ°Ñ Ð·Ð°Ð¿ÑÑÐºÐ° Ð¿Ð¾ÑÐ¾ÐºÐ¾Ð² â9 |

**Исходный код программы**

|  |
| --- |
| #include "lab3.h" |
|  | #include <windows.h> |
|  | #include <iostream> |
|  | #include <stdio.h> |
|  | #include <wchar.h> |
|  |  |
|  |  |
|  | #define NUM\_THREADS 10 |
|  | #define MAX\_SEM\_COUNT 3 |
|  | #define NUM\_SYMB 4 |
|  |  |
|  | HANDLE SemA, SemB, SemC, SemD, SemE, SemF, SemG, SemH, SemI, SemK; |
|  | HANDLE hThread[NUM\_THREADS]; |
|  | HANDLE mutex; |
|  |  |
|  |  |
|  | DWORD WINAPI ThreadA(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | for (int i = 0; i < NUM\_SYMB; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'a'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | ReleaseSemaphore(SemA, 1, NULL); |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadB(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | for (int i = 0; i < NUM\_SYMB\*2; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'b'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | ReleaseSemaphore(SemB, 1, NULL); |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadC(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | for (int i = 0; i < NUM\_SYMB\*3; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'c'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | ReleaseSemaphore(SemC, 3, NULL); |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadD(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | WaitForSingleObject(SemA, INFINITE); |
|  | for (int i = 0; i < NUM\_SYMB; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'd'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | ReleaseSemaphore(SemD, 1, NULL); |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadE(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | WaitForSingleObject(SemD, INFINITE); |
|  | WaitForSingleObject(SemB, INFINITE); |
|  | for (int i = 0; i < NUM\_SYMB; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'e'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | ReleaseSemaphore(SemE, 3, NULL); |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadF(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | WaitForSingleObject(SemC, INFINITE); |
|  | WaitForSingleObject(SemE, INFINITE); |
|  | for (int i = 0; i < NUM\_SYMB; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'f'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | ReleaseSemaphore(SemF, 1, NULL); |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadG(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | WaitForSingleObject(SemF, INFINITE); |
|  | for (int i = 0; i < NUM\_SYMB; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'g'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | ReleaseSemaphore(SemG, 1, NULL); |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadH(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | WaitForSingleObject(SemC, INFINITE); |
|  | WaitForSingleObject(SemE, INFINITE); |
|  | for (int i = 0; i < NUM\_SYMB \* 2; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'h'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | ReleaseSemaphore(SemH, 1, NULL); |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadI(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | WaitForSingleObject(SemC, INFINITE); |
|  | WaitForSingleObject(SemE, INFINITE); |
|  | for (int i = 0; i < NUM\_SYMB \* 3; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'i'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | return TRUE; |
|  | } |
|  |  |
|  | DWORD WINAPI ThreadK(LPVOID lpParam) { |
|  | UNREFERENCED\_PARAMETER(lpParam); |
|  | WaitForSingleObject(SemG, INFINITE); |
|  | WaitForSingleObject(SemH, INFINITE); |
|  | for (int i = 0; i < NUM\_SYMB; i++) { |
|  | WaitForSingleObject(mutex, INFINITE); |
|  | std::cout << 'k'; |
|  | ReleaseMutex(mutex); |
|  | Sleep(1); |
|  | } |
|  | return TRUE; |
|  | } |
|  |  |
|  |  |
|  | int lab3\_init() |
|  | { |
|  | mutex = CreateMutex(NULL, FALSE, NULL); |
|  |  |
|  | SemA = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemA == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemB = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemB == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemC = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemC == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemD = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemD == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemE = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemE == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemF = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemF == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemG = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemG == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemH = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemH == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemI = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemI == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | SemK = CreateSemaphore(NULL, 0, MAX\_SEM\_COUNT, NULL); |
|  | if (SemK == NULL) { printf("CreateSemaphore error: %d\n", GetLastError()); return 1; } |
|  |  |
|  | DWORD ThreadID; |
|  | int i = 0; |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadA, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadB, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadC, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadD, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadE, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadF, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadG, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadH, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadI, NULL, 0, &ThreadID); |
|  | hThread[i++] = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadK, NULL, 0, &ThreadID); |
|  |  |
|  | WaitForMultipleObjects(NUM\_THREADS, hThread, TRUE, INFINITE); |
|  | for (int i = 0; i < NUM\_THREADS; i++) |
|  | { |
|  | CloseHandle(hThread[i]); |
|  | } |
|  | CloseHandle(mutex); |
|  | CloseHandle(SemA); |
|  | CloseHandle(SemB); |
|  | CloseHandle(SemC); |
|  | CloseHandle(SemD); |
|  | CloseHandle(SemE); |
|  | CloseHandle(SemF); |
|  | CloseHandle(SemG); |
|  | CloseHandle(SemH); |
|  | CloseHandle(SemI); |
|  | CloseHandle(SemK); |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | unsigned int lab3\_task\_number() |
|  | { |
|  | return 9; |
|  | } |

**Выводы**

В соответствии с заданным графом запуска потоков реализован последовательно-параллельный запуск потоков. Последовательное выполнение потоков обеспечивается за счет использования семафоров. При выполнении операций ввода-вывода перед входом в критическую секцию потоки захватывают мьютекс.