БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий

и робототехники

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

О Т Ч Ё Т

о лабораторной работе № 1

Дисциплина

«Системное программирование»

Тема

«Синхронизация потоков в Windows»

Выполнил: студент гр. 10702217 Храмков Д. С.

Проверил: Разорёнов Н. А.

Минск 2019

***Лабораторная работа № 2***

**СИНХРОНИЗАЦИЯ ПОТОКОВ В WINDOWS**

**Цель работы:** изучить основы синхронизации потоков в ОС Windows

**Изучаемые вопросы**

1. Блокирующие функции. Защищённый доступп к переменным.
2. Критические секции.
3. Ожидающие функции.
4. Взаимоисключения.
5. События.
6. Семафоры.

***Постановка задачи***

Разработать многопоточное Win32-приложение, синхронизирующее работу трёх вторичных потоков:

* первый помещает данные типа Type в хранилище на n элементов;
* второй сортирует данные в хранилище;
* третий извлекает данные из хранилища и визуализирует их.

1. **Блокирующие функции. Защищённый доступ к переменным.**

В качестве блокирующих были рассмотрены атомарные функции. Их назначение не давать нескольким источникам (потокам) захватывать один ресурс.

InterlockedExchangeAdd(&var1, 1);

1. **Критические секции.**

Критические секции обеспечивают синхронизацию потоков путём ограничения доступа нескольким потокам к одному ресурсу, т.е. в одну единицу времени ресурсом может пользоваться только один поток.

EnterCriticalSection(&critSect);

for (int i = 0; i < 5; i++) {

Sleep(10);

var2[i] = i \* 2 + 1;

}

LeaveCriticalSection(&critSect);

1. **Ожидающие функции.**

Ожидающие функции останавливают работу других потоков, если указанный в них поток не выполнен.

WaitForSingleObject(hThread[0], INFINITE);

WaitForMultipleObjects(2, hThread, TRUE, INFINITE);

1. **Взаимоисключения.**

Взаимоисключения обеспечиваются использованием мьютексов – объектов ядра, гарантирующих взаимоисключающий доступ потоков к объекту. Мьютекс даже позволяет синхронизировать потоки из разных процессов.

hMutex = CreateMutex(NULL, FALSE, L"Individual");

WaitForSingleObject(OpenMutex(NULL, FALSE, L"Individual"), INFINITE);

if (a == 10) {

a = 0;

}

for (int i = 0; i < 5; i++) {

var4[a] = buffer[i];

a++;

Sleep(10);

}

ReleaseMutex(hMutex);

1. **События.**

Объекты события регулируют ход выполнения программы путём проверки флага состояния (свободен или занят).

WaitForSingleObject(hEvent, INFINITE);

for (int i = 0; i < 5; i++) {

var2[i] = i \* 2 + 1;

}

return 0;

1. **Семафоры.**

Семафоры регулируют доступ потоков к ресурсу при помощи счётчика процессов, получивших доступ к ресурсу, сравнивая его значение с максимальным допустимым количеством потоков. Как и мьютексы, семафоры являются объектами ядра.

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

WaitForSingleObject(hSemaphore, INFINITE);

if (a >= 10) {

a = 0;

}

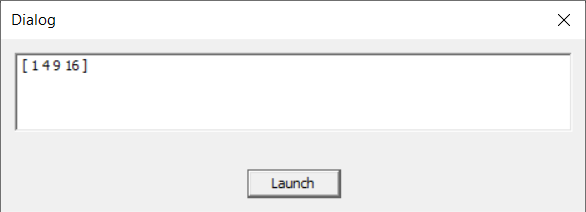
var6[a] = (a+1) \* (a+1);

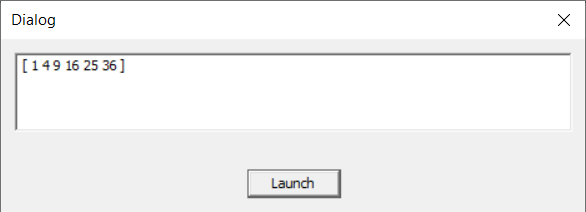
Sleep(100);

a++;

ReleaseSemaphore(hSemaphore, 1, NULL);

**Результаты выполнения программы:**





**Выводы:**

1. **Изучили синхронизацию потоков путём блокировки ресурса;**
2. **Изучили синхронизацию потоков путём блокировки доступа к сниппету;**
3. **Научились синхронизировать потоки в пользовательском режиме;**
4. **Научились синхронизировать потоки с использованием объекта ядра.**