Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧЕТ**

по лабораторной работе №2

по дисциплине «Основы операционных систем»

на тему «Синхронизация нитей с использованием событий и

Interlocked-функций»

Выполнили студенты группы 22ВВП1

Демин М. С.

Беляев Д. И.

Приняли:

Егоров В. Ю.

Федюнин Р. Н.

Пенза 2024

**Название**

Синхронизация нитей с использованием событий и Interlocked-функций

**Цель работы**

Изучение способа организации критических секций с использованием

группы Interlocked-функций и структур CRITICAL\_SECTION. Изучение

способа синхронизации процессов и нитей с использованием событий

(Events).

**Лабораторное задание**

Имеется 9 нитей. Каждая нить в цикле выводит на экран свой

уникальный символ. Вывод символа на экран производится из

критической секции на основе Interlocked-функций.

**Описание программы**

Данная программа написана на языке C и использует многопоточность для работы с общей переменной. Основная цель программы — продемонстрировать использование потоков и синхронизацию доступа к общей переменной с помощью атомарных операций.

**Структура программы**

Константы:

* #define THREAD\_COUNT 9: определяет количество потоков, которые будут созданы.

Глобальные переменный:

* static LONG volatile STATUS = 0;: используется для управления состоянием критической секции.
* static int SHARED\_DATA = 0;: общая переменная, к которой обращаются потоки.

Структура данных:

typedef struct {

int number;

} ThreadData;

* ThreadData: структура, содержащая номер потока, который будет передан каждому потоку при его создании.

Функция потока:

DWORD WINAPI WorkThread(void\* p)

* Эта функция выполняется каждым потоком.
* Входной параметр p — указатель на структуру ThreadData.
* Используется InterlockedExchange для управления доступом к общей переменной SHARED\_DATA, что предотвращает одновременное изменение этой переменной несколькими потоками.
* После получения доступа к критической секции поток выводит свой номер и текущее значение SHARED\_DATA, затем увеличивает это значение на 1.
* В конце функция освобождает память, выделенную под ThreadData.

Главная функция main():

* Создает массив дескрипторов потоков.
* В цикле создает THREAD\_COUNT потоков, каждому из которых передает уникальный номер через структуру ThreadData.
* Проверяет успешность создания каждого потока. Если создание не удалось, программа выводит сообщение об ошибке и завершает выполнение.
* После создания всех потоков вызывается WaitForMultipleObjects, чтобы дождаться завершения всех потоков перед выходом из программы.

**Общее описание работы программы**

Программа создает 9 потоков, каждый из которых получает свой уникальный номер. Потоки конкурируют за доступ к общей переменной SHARED\_DATA, используя атомарные операции для предотвращения гонок данных. Каждый поток выводит свой номер и текущее значение SHARED\_DATA, затем увеличивает это значение на 1. Программа завершается после завершения всех потоков.

**Текст программы**

#include <stdio.h>

#include <windows.h>

#include <time.h>

#define THREAD\_COUNT 9

static LONG volatile STATUS = 0;

static int SHARED\_DATA = 0;

typedef struct {

int number;

} ThreadData;

DWORD WINAPI WorkThread(void\* p)

{

ThreadData\* data = (ThreadData\*)p;

while (TRUE) {

// Start CRITICAL\_SECTION

while (InterlockedExchange(&STATUS, TRUE) == TRUE) {

Sleep(100);

}

wprintf(L"%d work. Shared data: %d\n", data->number, SHARED\_DATA);

SHARED\_DATA++;

InterlockedExchange(&STATUS, FALSE);

// End CRITICAL\_SECTION

Sleep(20);

}

free(data);

return 0;

}

int main()

{

HANDLE threads[THREAD\_COUNT];

for (size\_t i = 0; i < THREAD\_COUNT; i++) {

ThreadData\* data = (ThreadData\*)malloc(sizeof(ThreadData));

data->number = i;

HANDLE thread = CreateThread(NULL, 0, WorkThread, data, 0, NULL);

if (thread == NULL) {

wprintf(L"CreateThread failed (%d).\n", GetLastError());

exit(EXIT\_FAILURE);

}

threads[i] = thread;

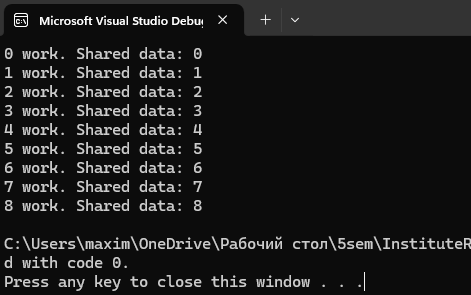
}

WaitForMultipleObjects(THREAD\_COUNT, threads, TRUE, INFINITE);

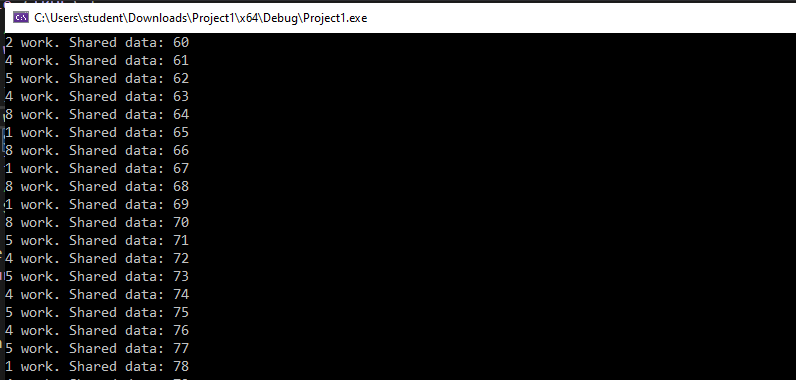
return 0;

}

**Результат работы программы**

****

При использовании функции InterlockedExchange важно помнить, что освобождение информации происходит не моментально и нужно жертвовать скоростью работы, иначе велик шанс, что вся работа будет выполняться только в одной нити.



**Вывод**

Изучили способ организации критических секций с использованием

группы Interlocked-функций и структур CRITICAL\_SECTION. Изучили

способ синхронизации процессов и нитей с использованием событий (Events).