Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №4

на тему

ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ: ВЗАИМНОЕ ИСКЛЮЧЕНИЕ И СИНХРОНИЗАЦИЯ

Выполнил: студент гр.253505 Косяков М.М.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178093096)

[2 Краткие теоритические сведения 4](#_Toc178093097)

[3 Описание функций программы 5](#_Toc178093098)

[3.1 Ввод количества читателей и писателей 5](#_Toc178093099)

[3.2 Обработка данных 5](#_Toc178093100)

[Заключение 7](#_Toc178093101)

[Список использованных источников 8](#_Toc178093102)

[Приложение А (обязательное) Исходный код программы 9](#_Toc178093103)

1 ФОРМУЛИРОВКА ЗАДАЧИ

В рамках этой лабораторной работы ставлю перед собой цель углубить и закрепить навыки работы с подходами, системными объектами и функциями для обеспечения синхронизации и передачи управления между взаимодействующими процессами в операционной системе Windows. В ходе выполнения работы необходимо изучить типичные проблемы, возникающие при организации взаимодействия процессов, модели для их описания и пути их решения. Особое внимание будет уделено задаче взаимного исключения и синхронизации, которые являются одними из основных задач взаимодействия процессов. Также предстоит изучить работу с методами предотвращения некорректного совместного доступа к критическим ресурсам.

Для выполнения лабораторной работы по созданию модели «писатели-читатели»необходимо реализовать приложение, которое будет работать с потоками двух видов. Также оно будет демонстрировать работу многозадачного комплекса. Программа будет написана на языке C с использованием WinAPI.

В качестве задачи необходимо выполнить моделирование распределенной обработки, а для этого:

– ввести количество читателей;

– ввести количество писателей;

– реализовать потоки читателей, которые постоянно будут пытаться прочитать общий ресурс, соблюдая механизмы синхронизации;

– реализовать потоки писателей, которые постоянно будут пытаться писать в общий ресурс, соблюдая правила потокобезопасности;

В результате выполнения этой лабораторной работы будут не только получены теоретические знания работы с методами обеспечения передачи данных между взаимодействующими процессами, но и практический опыт в разработке приложений, которые могут эффективно использовать ресурсы компьютера. Понимание концепций специализированных средств для обеспечения синхронизации и передачи управления между взаимодействующими процессами станет необходимым для создания высокопроизводительных программных решений.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Критический ресурс — ресурс системы, который не может использоваться одновременно более чем заданным числом пользователей [1]. Часто речь идет о доступности его не более чем одному пользователю.

Критическая секция — часть алгоритма, где происходят обращения к критическим ресурсам. Очевидно, число потоков, находящихся в критической секции, ограничивается в соответствии с характеристиками критического ресурса (часто — одним потоком). Требования к критическим секциям: ограничение количества потоков, ограничение времени пребывания, отсутствие приоритетов внутри секции, независимость от конкретных параметров системы.

Проблема взаимного исключения заключается в исключении выполнения действий, которые не должны происходить одновременно, например, при выводе документов на печать, где данные должны быть обработаны последовательно.

Проблема синхронизации заключается в обеспечении правильного порядка выполнения действий между потоками или процессами для корректного взаимодействия и передачи управления между ними [2].

Проблема обмена данными вытекает из концепции изолированных адресных пространств — процессы не имеют возможности напрямую обращаться «внутрь» друг друга. Потоки одного процесса действуют в едином адресном пространстве, но в общем случае во взаимодействии могут участвовать потоки разных процессов.

Решением всех задач становится использование специальных системных объектов — объектов IPC (Inter-Process Communication) и, как частный случай, ISO (Inter-process Synchronization Objects).

Механизмы синхронизации используются для управления доступом к общим ресурсам. Основная идея заключается в проверке и изменении флага, отражающего состояние ресурса (свободен или занят). Однако сами механизмы контроля доступа могут становится критическим ресурсом и вызвать зацикливание требований.

Для обеспечения атомарности операций с критическими секциями используются привилегированные операции, недоступные обычным процессам.

Мьютекс— объект для обеспечения взаимного исключения, используется для контроля доступа к критическому ресурсу. Механизм Critical Section — упрощенный мьютекс для реализации критической секции в рамках одного процесса. Семафор—глобальная переменная-счётчик, которая позволяет управлять доступом к ресурсу с помощью двух операций.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы [3]:

– ввод количества читателей;

– ввод количества писателей;

– вывод результата обработки;

– завершение работы программы;

**3.1 Ввод количества читателей и писателей**

Для начала обработки следует ввести количество читателей, для того чтобы в будущем создать нужное количество потоков читателей (рисунок 3.1).

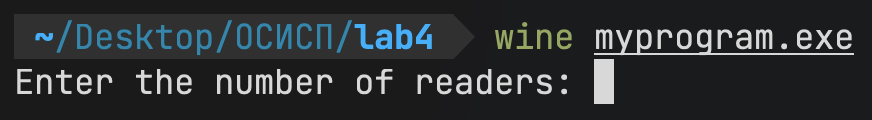


Рисунок 3.1 – Ввод количества читателей

Далее следует ввести количество писателей, для того чтобы в будущем создать нужное количество потоков писателей (рисунок 3.2).

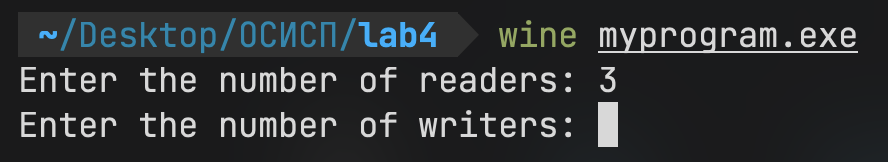


Рисунок 3.2 – Ввод количества писателей

**3.2 Обработка данных**

Программа реализует многопоточное взаимодействие в Windows с использованием именованных синхронизационных объектов. В программе присутствуют два типа потоков: читатели и писатели. Читатели постоянно пытаются прочитать общий ресурс, соблюдая механизмы синхронизации. Писатели также непрерывно пытаются писать в общий ресурс, соблюдая правила потокобезопасности. Программа отслеживает количество завершенных потоков-читателей и завершает работу всего процесса, когда определенное количество читателей завершили свои задачи. Для синхронизации доступа к общему ресурсу используются именованные семафоры и мьютексы. Основная функция создает потоки читателей и писателей, дожидается завершения их работы, освобождает ресурсы и завершает выполнение программы. Этот подход обеспечивает безопасное взаимодействие между потоками и контроль доступа к общему ресурсу в многопоточной среде операционной системы Windows (рисунок 3.3).

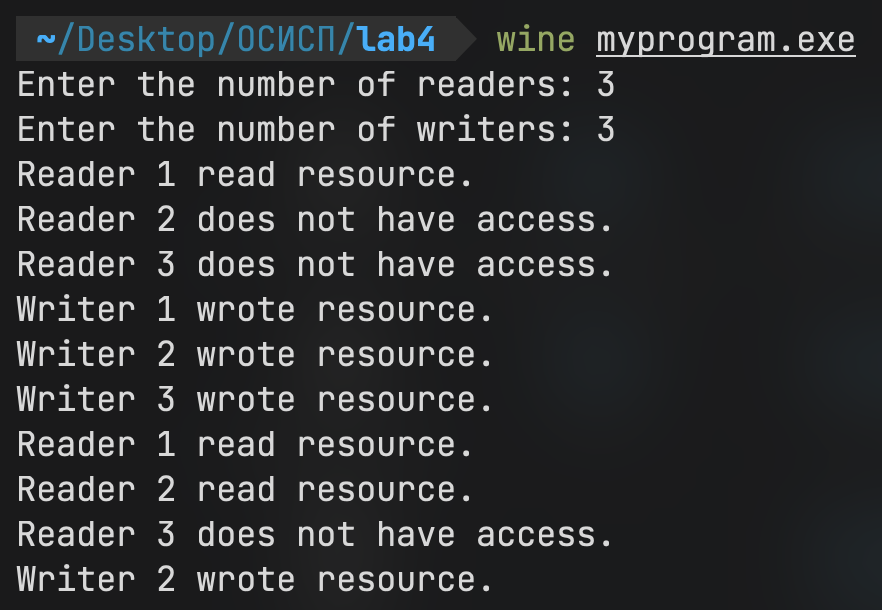


Рисунок 3.3 – Сообщение об обработке

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены и закреплены навыки программирования приложений для операционной системы Windows с использованием подходов, системных объектов и функций для обеспечения синхронизации и передачи управления между взаимодействующими процессами. Мы погрузились в задачи взаимного исключения и синхронизации, которые являются одними из задач взаимодействия процессов, и научились их реализовывать в среде Windows с помощью WinAPI.

Основными аспектами, которыми мы ознакомились, стали критический ресурс и критическая секция, проблема атомарности доступа для проверки и модификации глобальных объектов, используемых для синхронизации. Мы также освоили типичные проблемы, возникающие при организации взаимодействия процессов, модели для их описания и пути их решения, что сыграет важную роль в разработке производительных приложений.

В ходе работы над созданием модели «писатели-читатели»мы реализовали приложение, которое работаем с потоками двух видов. Также оно демонстрирует работу многозадачного комплекса.

Эта лабораторная работа не только позволила углубить наши теоретические знания в области взаимодействия процессов, но и предоставила ценный практический опыт в разработке эффективных приложений, способных эффективно использовать ресурсы компьютера. Понимание концепций взаимного исключения и синхронизации, а также навыки управления, стали ключевыми в создании высокопроизводительных программных решений.

Выполнение этой лабораторной работы позволило нам не только расширить нашу базу знаний, но и приобрести практические навыки, которые будут полезны в дальнейшей разработке программного обеспечения для Windows.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/

[2] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

[3] Разработка приложений с помощью WinAPI. – Режим доступа: https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <stdio.h>

#include <Windows.h>

struct ThreadInfo

{

int id;

HANDLE threadHandle;

};

struct SharedData

{

HANDLE semaphore; // Semaphore for controlling access to the shared resource

HANDLE outputMtx; // Mutex for thread-safe output

int resource; // Shared resource

int readersCompleted; // Number of completed readers

};

SharedData sharedData;

// Reader thread function

DWORD WINAPI ReaderThread(LPVOID arg)

{

ThreadInfo \*threadInfo = (ThreadInfo \*)arg;

int id = threadInfo->id;

while (1)

{

DWORD result = WaitForSingleObject(sharedData.semaphore, 0); // Try to acquire the semaphore

InterlockedIncrement((LONG \*)&sharedData.readersCompleted); // Increment the number of completed readers

if (sharedData.readersCompleted >= 7) // Check if 7 readers have completed

{

ExitProcess(0); // Terminate the entire process

}

if (result == WAIT\_OBJECT\_0) // If semaphore acquired

{

WaitForSingleObject(sharedData.outputMtx, INFINITE); // Acquire output mutex for thread-safe output

printf("Reader %d read resource.\n", id);

ReleaseMutex(sharedData.outputMtx); // Release output mutex

ReleaseSemaphore(sharedData.semaphore, 1, NULL); // Release semaphore after reading

}

else if (result == WAIT\_TIMEOUT) // If semaphore not acquired

{

WaitForSingleObject(sharedData.outputMtx, INFINITE); // Acquire output mutex for thread-safe output

printf("Reader %d does not have access.\n", id);

ReleaseMutex(sharedData.outputMtx); // Release output mutex

}

Sleep(500); // Emulate reading time

}

return 0;

}

// Writer thread function

DWORD WINAPI WriterThread(LPVOID arg)

{

ThreadInfo \*threadInfo = (ThreadInfo \*)arg;

int id = threadInfo->id;

while (1)

{

WaitForSingleObject(sharedData.semaphore, INFINITE); // Acquire semaphore for writing

sharedData.resource = id; // Write to the shared resource

WaitForSingleObject(sharedData.outputMtx, INFINITE); // Acquire output mutex for thread-safe output

printf("Writer %d wrote resource.\n", id); // Write to the resource

ReleaseMutex(sharedData.outputMtx); // Release output mutex

ReleaseSemaphore(sharedData.semaphore, 1, NULL); // Release semaphore after writing

Sleep(1000); // Emulate writing time

}

return 0;

}

int main()

{

sharedData.semaphore = CreateSemaphore(NULL, 1, 1, NULL); // Create semaphore with initial count of 1

sharedData.outputMtx = CreateMutex(NULL, FALSE, NULL); // Create mutex for thread-safe output

sharedData.resource = 0;

int readersNum, writersNum;

printf("Enter the number of readers: ");

scanf("%d", &readersNum);

printf("Enter the number of writers: ");

scanf("%d", &writersNum);

ThreadInfo \*readerThreads = (ThreadInfo \*)malloc(readersNum \* sizeof(ThreadInfo));

ThreadInfo \*writerThreads = (ThreadInfo \*)malloc(writersNum \* sizeof(ThreadInfo));

HANDLE \*readerHandles = (HANDLE \*)malloc(readersNum \* sizeof(HANDLE));

HANDLE \*writerHandles = (HANDLE \*)malloc(writersNum \* sizeof(HANDLE));

for (int i = 0; i < readersNum; i++)

{

readerThreads[i].id = i + 1;

readerThreads[i].threadHandle = CreateThread(NULL, 0, ReaderThread, &readerThreads[i], 0, NULL);

readerHandles[i] = readerThreads[i].threadHandle;

}

for (int i = 0; i < writersNum; i++)

{

writerThreads[i].id = i + 1;

writerThreads[i].threadHandle = CreateThread(NULL, 0, WriterThread, &writerThreads[i], 0, NULL);

writerHandles[i] = writerThreads[i].threadHandle;

}

WaitForMultipleObjects(readersNum, readerHandles, TRUE, INFINITE);

WaitForMultipleObjects(writersNum, writerHandles, TRUE, INFINITE);

CloseHandle(sharedData.semaphore);

CloseHandle(sharedData.outputMtx);

free(readerThreads);

free(writerThreads);

free(readerHandles);

free(writerHandles);

return 0;

}