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 №3

на тему

ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ: ОБМЕН ДАННЫМИ

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 текста для передачи 5](#_Toc178093099)

[3.2 Обработка данных 5](#_Toc178093100)

[Заключение 6](#_Toc178093101)

[Список использованных источников 7](#_Toc178093102)

[Приложение А (обязательное) Исходный код программы 8](#_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 передачи данных между взаимодействующими процессами и совместной их обработки в операционной системе Windows. В ходе выполнения работы необходимо изучить типичные проблемы, возникающие при организации взаимодействия процессов, и пути их решения. Особое внимание будет уделено задаче обмена данными, которая является одной из задач взаимодействия процессов. Также предстоит изучить работу со специализированными средствами IPC для обмена данными, разобрать преодоление изоляции процессов в многозадачной системе.

Для выполнения лабораторной работы по моделированию распределенной обработки необходимо реализовать приложение, которое будет работать с потоками двух видов. Также оно будет демонстрировать работу многозадачного комплекса с обменом. Программа будет написана на языке C с использованием WinAPI.

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

– ввести текст для передачи в другой поток;

– реализовать поток диспетчер, который будет создавать объект IPC, выдавать задание рабочему потоку, осуществлять сбор результатов;

– реализовать рабочий поток, который будет принимать задание, выполнять обработку и возвращать результат;

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 IPC для обмена данными 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.

Проблема обмена данными вытекает из концепции изолированных адресных пространств — процессы не имеют возможности напрямую обращаться «внутрь» друг друга [2]. 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).

IPC — это способ обмена данными между потоками внутри одного процесса или между потоками разных процессов. Существует несколько способов классификации IPC: сигнальные, канальные, средства синхронизации, разделяемая память, очереди сообщений и комбинированные решения.

Каналы — это объекты, которые работают аналогично файлам и обеспечивают межпроцессное взаимодействие.

Сокеты — это механизмы взаимодействия, близкиие к каналам, используются в сетевом и локальном программировании.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

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

– ввод текста для передачи в другой поток;

– вывод обработанного текста;

– завершение работы программы;

**3.1 Ввод текста для передачи**

Для начала обработки следует ввести текст (рисунок 3.1).

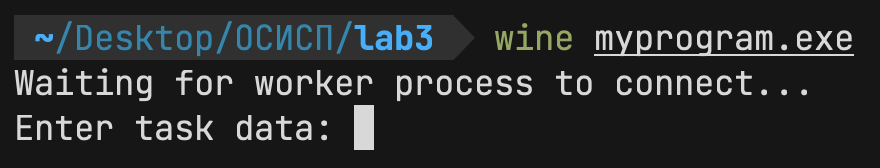


Рисунок 3.1– Ввод текста для передачи

**3.2 Обработка данных**

Программа реализует взаимодействие между процессами через именованные каналы. Она содержит функции для диспетчера и рабочего процесса. Диспетчер отправляет задачу рабочему процессу через именованный канал, а рабочий процесс выполняет операцию XOR над данными и отправляет результат обратно. Диспетчер выводит результат (рисунок 3.2).

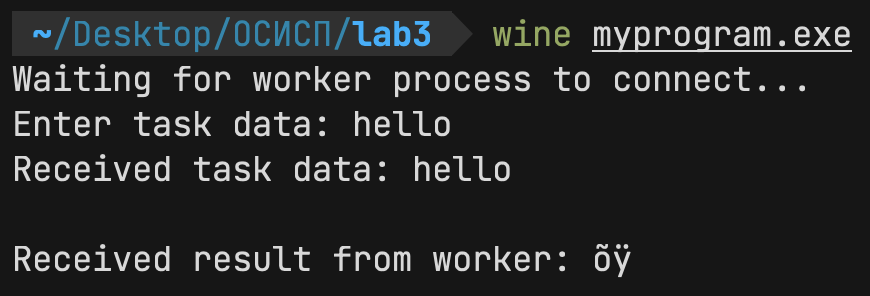


Рисунок 3.2 – Сообщение об обработке данных

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 передачи данных между взаимодействующими процессами и совместной их обработки. Мы погрузились в задачу обмена данными, которая является одной из задач взаимодействия процессов, и научились её реализовывать в среде Windows с помощью WinAPI.

Основными аспектами, которыми мы ознакомились, стали специализированные средства IPC для обмена данными, методы преодоления изоляции процессов в многозадачной системе. Мы также освоили типичные проблемы, возникающие при организации взаимодействия, и пути их решения, что сыграет важную роль в разработке производительных приложений.

В ходе работы над моделированием распределенной обработки мы реализовали приложение, которое работаем с потоками двух видов. Также оно демонстрирует работу многозадачного комплекса с обменом.

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 обмена данными между процессами, а также навыки их 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 <windows.h>

#include <stdio.h>

#define PIPE\_NAME "\\\\.\\pipe\\XORPipe"

#define BUFFER\_SIZE 1024

// One channel for issuing tasks and one channel for receiving for all worker processes

void xor\_process(char \**data*, int *len*)

{

for (int i = 0; i < *len*; i++)

{

*data*[i] ^= 0xFF;

}

}

// Creates the named pipe, sends a task to the worker process, and receives the processed result

DWORD WINAPI DispatcherProcess(LPVOID *lpParam*)

{

HANDLE hPipe;

char buffer[BUFFER\_SIZE];

DWORD dwWritten, dwRead;

// Create a named pipe

hPipe = CreateNamedPipeA(

PIPE\_NAME, PIPE\_ACCESS\_DUPLEX, PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,

PIPE\_UNLIMITED\_INSTANCES, BUFFER\_SIZE, BUFFER\_SIZE, 0, NULL);

if (hPipe == INVALID\_HANDLE\_VALUE)

{

printf("Error creating named pipe: %d\n", GetLastError());

return 1;

}

printf("Waiting for worker process to connect...\n");

// Retry loop for connection

BOOL isConnected = ConnectNamedPipe(hPipe, NULL) ? TRUE : (GetLastError() == ERROR\_PIPE\_CONNECTED);

if (!isConnected)

{

printf("Error connecting named pipe: %d\n", GetLastError());

CloseHandle(*hPipe*);

return 1;

}

char taskData[BUFFER\_SIZE];

printf("Enter task data: ");

fgets(*taskData*, BUFFER\_SIZE, stdin);

// Send task to worker process

if (!WriteFile(hPipe, taskData, strlen(taskData) + 1, &dwWritten, NULL))

{

printf("Failed to send data to worker: %d\n", GetLastError());

DisconnectNamedPipe(hPipe);

CloseHandle(hPipe);

return 1;

}

// Receive result from worker process

if (!ReadFile(hPipe, buffer, BUFFER\_SIZE, &dwRead, NULL))

{

printf("Failed to read from worker: %d\n", GetLastError());

DisconnectNamedPipe(hPipe);

CloseHandle(hPipe);

return 1;

}

printf("Received result from worker: %s\n", buffer);

// Clean up

DisconnectNamedPipe(*hPipe*);

CloseHandle(*hPipe*);

return 0;

}

// Connects to the named pipe, receives the task, performs the XOR processing, and sends the result back.

DWORD WINAPI WorkerProcess(LPVOID *lpParam*)

{

HANDLE hPipe;

char buffer[BUFFER\_SIZE];

DWORD dwWritten, dwRead;

// Retry loop for connection

while (1)

{

hPipe = CreateFileA(

PIPE\_NAME, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, 0, NULL);

if (hPipe != INVALID\_HANDLE\_VALUE)

break;

if (GetLastError() != ERROR\_PIPE\_BUSY)

{

printf("Could not open pipe: %d\n", GetLastError());

return 1;

}

// Wait for the pipe to become available

if (!WaitNamedPipeA(PIPE\_NAME, 2000)) // Wait 2000 milliseconds before try again to connect

{

printf("Could not open pipe within wait period\n");

return 1;

}

}

// Receive task data

if (!ReadFile(hPipe, buffer, BUFFER\_SIZE, &dwRead, NULL))

{

printf("Failed to read from dispatcher: %d\n", GetLastError());

CloseHandle(hPipe);

return 1;

}

printf("Received task data: %s\n", buffer);

// Process data (XOR operation)

xor\_process(*buffer*, *dwRead*);

// Send processed data back

if (!WriteFile(hPipe, buffer, dwRead, &dwWritten, NULL))

{

printf("Failed to send result to dispatcher: %d\n", GetLastError());

CloseHandle(hPipe);

return 1;

}

CloseHandle(hPipe);

return 0;

}

int main()

{

HANDLE hDispatcher, hWorker;

// Create dispatcher and worker threads

hDispatcher = CreateThread(NULL, 0, DispatcherProcess, NULL, 0, NULL);

hWorker = CreateThread(NULL, 0, WorkerProcess, NULL, 0, NULL);

WaitForSingleObject(hDispatcher, INFINITE);

WaitForSingleObject(hWorker, INFINITE);

CloseHandle(hDispatcher);

CloseHandle(hWorker);

return 0;

}