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 Павлович В.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc182226538)

[2 Краткие теоретические сведения 4](#_Toc182226539)

[3 Описание работы программы 5](#_Toc182226540)

[3.1 Клиентское приложение 5](#_Toc182226541)

[3.2 Серверное приложение 5](#_Toc182226542)

[Заключение 7](#_Toc182226543)

[Список использованных источников 8](#_Toc182226544)

[Приложение А (Обязательное) Исходный код программы 9](#_Toc182226545)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью лабораторной работы является изучение Подходы, системные объектов и функций для обеспечения передачи данных между взаимодействующими процессами и/или совместной их обработки, а также типичных проблем, возникающих при организации взаимодействия, и пути их решения.

В качестве задачи необходимо разработать приложение, реализующее многопользовательский (с мультиплексированием) вывод. Процесс-сервер ожидает данных (сообщений) от нескольких источников (клиентов), записывает сообщения в файл с соблюдением порядка поступления, снабжая дополнительной информацией (временная метка, идентификатор источника и т.п.) и с соблюдением определенного формата.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Операционная система Windows предоставляет механизмы для упрощения обмена данными и обмена данными между приложениями. В совокупности действия, включенные этими механизмами, называются межпроцессными коммуникациями (IPC). Некоторые формы IPC упрощают разделение труда между несколькими специализированными процессами. Другие формы IPC упрощают разделение труда между компьютерами в сети.

Как правило, приложения могут использовать IPC, классифицированные как клиенты или серверы. Клиент – это приложение или процесс, который запрашивает службу от другого приложения или процесса. Сервер – это приложение или процесс, который отвечает на запрос клиента. Многие приложения действуют как клиент, так и сервер в зависимости от ситуации.[1]

Одним из механизмов межпроцессной коммуникации являются каналы. Именованный канал – это именованный односторонний или дуплексный канал для обмена данными между сервером канала и одним или несколькими клиентами канала. Все экземпляры именованного канала имеют одно и то же имя канала, но каждый экземпляр имеет собственные буферы и дескрипторы и предоставляет отдельный канал для обмена данными между клиентом и сервером. Использование экземпляров позволяет нескольким клиентам канала одновременно использовать один и тот же именованный канал.

Любой процесс может получить доступ к именованным каналам при проверке безопасности, что делает именованные каналы легкой формой взаимодействия между связанными или несвязанными процессами.

Любой процесс может выступать как сервер, так и клиент, что делает возможным одноранговый обмен данными. Как используется здесь, термин сервер канала относится к процессу, который создает именованный канал, а термин "клиент канала" относится к процессу, который подключается к экземпляру именованного канала. Серверная функция для создания экземпляра именованного канала – CreateNamedPipe. Серверная функция для принятия подключения – ConnectNamedPipe. Клиентский процесс подключается к именованной трубе с помощью функции CreateFile или CallNamedPipe.

Именованные каналы можно использовать для обеспечения взаимодействия между процессами на одном компьютере или между процессами на разных компьютерах по сети. Если служба сервера запущена, все именованные каналы доступны удаленно.[2]

3 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

В этом разделе будут рассмотрены основные функции программ, а также механизмы их взаимодействия.

## **3.1 Клиентское приложение**

Вся функциональность клиентского приложения заключается в получении текста с консоли с последующей записью его в именованный канал. После получения ввода с консоли, клиентское приложение подключается к именованному каналу с использованием функции CreateFile, после чего записывает в этот канал полученные данные и ожидает получения новых данных из консоли. Пользовательский интерфейс клиентского приложения представлен на рисунке 3.1.

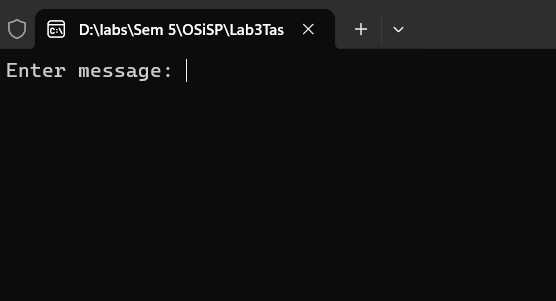


Рисунок 3.1 – Пользовательский интерфейс клиентского приложения

## **3.2 Серверное приложение**

Серверное приложение фиксирует подключение каждого клиентского приложения к именованному каналу и выделяет поток для обработки этого подключения, выводит информацию о подключении в консоль. Далее, при записи клиентом информации в канал, сервер считывает эту информацию и записывает в файл логов дату, идентификатор пользователя и полученные данные. На рисунке 3.2 представлен пользовательский интерфейс серверного приложения, на рисунке 3.3 представлено содержимое файла с логами после нескольких операций со стороны клиентов.

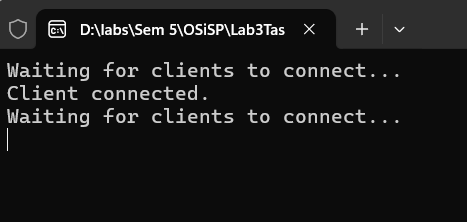


Рисунок 3.2 – Пользовательский интерфейс серверного приложения

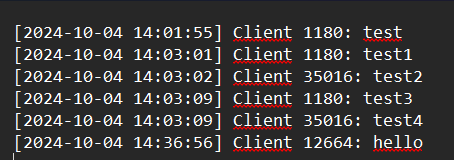


Рисунок 3.3 – Содержимое файла с логами

Таким образом, разработанное приложение демонстрирует возможности межпроцессного взаимодействия при помощи именованного канала.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы были изучены подходы и методы, необходимые для организации передачи данных между взаимодействующими процессами.

В результате были реализованы 2 приложения: клиент и сервер, демонстрирующие мультиплексирование вывода в файл информации, полученной из нескольких источников с сохранением порядка получения данных.

Для обмена данными между процессами использовались именованные каналы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Взаимодействие между процессами – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/ipc/interprocess-communications.

[2] Именованные каналы – Win32 apps [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/ipc/named-pipes.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

Листинг А1 – Исходный код клиентского приложения

#include <windows.h>

#include <stdio.h>

#define BUFFER\_SIZE 1024

#define PIPE\_NAME L"\\\\.\\pipe\\LogPipe"

int main() {

HANDLE pipe;

char buffer[BUFFER\_SIZE];

DWORD bytesWritten;

while (1) {

pipe = CreateFile(

PIPE\_NAME,

GENERIC\_WRITE,

0,

NULL,

OPEN\_EXISTING,

0,

NULL);

if (pipe != INVALID\_HANDLE\_VALUE) {

break;

}

Sleep(1000);

}

while (1) {

printf("Enter message: ");

fgets(buffer, BUFFER\_SIZE, stdin);

buffer[strcspn(buffer, "\n")] = '\0'; // Remove newline character

if (!WriteFile(pipe, buffer, strlen(buffer) + 1, &bytesWritten, NULL)) {

fprintf(stderr, "Failed to write to pipe. Error: %ld\n", GetLastError());

break;

}

}

CloseHandle(pipe);

return 0;

}

Листинг А2 – Исходный код серверного приложения

#include <windows.h>

#include <stdio.h>

#include <time.h>

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#define BUFFER\_SIZE 1024

#define PIPE\_NAME L"\\\\.\\pipe\\LogPipe"

#define MAX\_CLIENTS 10

HANDLE logMutex;

void log\_message(const char\* client\_id, const char\* message) {

WaitForSingleObject(logMutex, INFINITE);

HANDLE file = CreateFile(

L"server\_log.txt",

FILE\_APPEND\_DATA,

FILE\_SHARE\_READ,

NULL,

OPEN\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

if (file != INVALID\_HANDLE\_VALUE) {

char log\_entry[BUFFER\_SIZE + 100];

time\_t now = time(NULL);

struct tm timeinfo;

localtime\_s(&timeinfo, &now);

char time\_str[20];

strftime(time\_str, sizeof(time\_str), "%Y-%m-%d %H:%M:%S", &timeinfo);

snprintf(log\_entry, sizeof(log\_entry), "[%s] Client %s: %s\n", time\_str, client\_id, message);

DWORD bytesWritten;

WriteFile(file, log\_entry, strlen(log\_entry), &bytesWritten, NULL);

CloseHandle(file);

}

else {

fprintf(stderr, "Failed to open log file. Error: %ld\n", GetLastError());

}

ReleaseMutex(logMutex);

}

DWORD WINAPI client\_handler(LPVOID param) {

HANDLE pipe = (HANDLE)param;

char buffer[BUFFER\_SIZE];

DWORD bytesRead;

char client\_id[10];

snprintf(client\_id, sizeof(client\_id), "%d", GetCurrentThreadId());

while (ReadFile(pipe, buffer, sizeof(buffer) - 1, &bytesRead, NULL)) {

buffer[bytesRead] = '\0';

log\_message(client\_id, buffer);

}

DisconnectNamedPipe(pipe);

CloseHandle(pipe);

return 0;

}

int main() {

HANDLE pipe;

HANDLE threads[MAX\_CLIENTS];

int threadCount = 0;

HANDLE hFile = CreateFile(

L"server\_log.txt",

GENERIC\_WRITE,

0,

NULL,

CREATE\_NEW,

FILE\_ATTRIBUTE\_NORMAL,

NULL);

CloseHandle(hFile);

logMutex = CreateMutex(NULL, FALSE, NULL);

if (logMutex == NULL) {

fprintf(stderr, "Failed to create mutex. Error: %ld\n", GetLastError());

return 1;

}

while (1) {

pipe = CreateNamedPipe(

PIPE\_NAME,

PIPE\_ACCESS\_INBOUND,

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,

MAX\_CLIENTS,

BUFFER\_SIZE,

BUFFER\_SIZE,

0,

NULL);

if (pipe == INVALID\_HANDLE\_VALUE) {

fprintf(stderr, "Failed to create named pipe. Error: %ld\n", GetLastError());

return 1;

}

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

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

if (connected) {

printf("Client connected.\n");

if (threadCount < MAX\_CLIENTS) {

threads[threadCount] = CreateThread(

NULL,

0,

client\_handler,

(LPVOID)pipe,

0,

NULL

);

threadCount++;

if (threads[threadCount] == NULL) {

printf("CreateThread failed, Error: %ld\n", GetLastError());

CloseHandle(pipe);

}

}

else {

fprintf(stderr, "Max clients reached. Connection refused.\n");

DisconnectNamedPipe(pipe);

CloseHandle(pipe);

}

}

}

WaitForMultipleObjects(threadCount, threads, TRUE, INFINITE);

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

CloseHandle(threads[i]);

}

CloseHandle(logMutex);

return 0;

}