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.253504 Новиков В.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178641666)

[2 Краткие теоритические сведения 3](#_Toc178641667)

[3 Описание функций программы 5](#_Toc178641668)

[Заключение 7](#_Toc178641669)

[Список использованных источников 8](#_Toc178641670)

[Приложение А (обязательное) исходный код продукта 9](#_Toc178641671)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является изучение механизмов межпроцессного взаимодействия (*IPC*) в системном программировании на примере реализации сервера логирования с поддержкой многопользовательского доступа. Цель работы заключается в демонстрации возможностей различных средств *IPC* для организации взаимодействия процессов и оценки их эффективности при одновременной обработке сообщений от нескольких источников.

В рамках задачи требуется разработать сервер логирования, который выполняет прием сообщений от процессов-клиентов, обрабатывает их и записывает в лог-файл с сохранением последовательности поступления. Для каждой записи в лог-файле необходимо добавлять дополнительную информацию, такую как временная метка, идентификатор источника и форматированные данные сообщения. Такой подход позволяет обеспечить точность и структурированность логирования в условиях многопроцессной среды.

Для реализации сервера предполагается использование механизмов *IPC*, таких как именованные или неименованные каналы, очереди сообщений или почтовые ящики. Эти средства обеспечивают передачу данных между процессами, исключая необходимость использования общей памяти, и позволяют организовать эффективную синхронизацию между сервером и клиентами. Особое внимание уделяется предотвращению коллизий при записи сообщений и сохранению их корректной последовательности.

Практическая часть включает тестирование разработанного сервера в условиях одновременной работы нескольких процессов-клиентов. Каждый клиент генерирует сообщения и передает их серверу для последующего протоколирования. Корректность реализации оценивается по отсутствию коллизий и правильной обработке сообщений, а эффективность — по времени выполнения операций и потребляемым ресурсам.

**2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ**

Операционная система *Windows* предоставляет широкий спектр механизмов, предназначенных для упрощения обмена данными и взаимодействия между приложениями. Эти механизмы, известные как межпроцессные коммуникации (*IPC*), играют ключевую роль в организации взаимодействия между процессами в одной системе или между различными устройствами в сети. Основной задачей *IPC* является обеспечение эффективной передачи данных, синхронизации и координации выполнения приложений.

Механизмы *IPC* предлагают два основных подхода к взаимодействию: упрощение разделения труда между несколькими процессами внутри одной системы и поддержка взаимодействия между удаленными компьютерами в сети. В первом случае *IPC* позволяет приложениям, работающим в одном окружении, обмениваться данными без прямого доступа к общей памяти, что особенно важно для многозадачных систем. Во втором случае *IPC* упрощает сетевое взаимодействие, позволяя распределенным приложениям обмениваться данными и координировать свои действия.[1]

Приложения, использующие *IPC*, обычно классифицируются как клиенты или серверы. Клиент — это приложение или процесс, который отправляет запрос на выполнение определенной операции или предоставление данных. Сервер, в свою очередь, — это приложение или процесс, который принимает запросы клиентов и выполняет их. Важно отметить, что многие современные приложения могут одновременно выступать как клиентами, так и серверами, в зависимости от контекста их работы.

Например, приложение текстового редактора может быть клиентом, отправляющим запрос на получение данных из приложения электронной таблицы, которое в данной ситуации выступает в роли сервера. Электронная таблица может предоставить сводную таблицу или другой аналитический отчет. Однако роль серверов и клиентов часто меняется: приложение электронной таблицы может само стать клиентом, запросив данные об уровне запасов у системы управления складом. Этот гибкий подход позволяет приложениям эффективно взаимодействовать в сложных программных экосистемах.

Таким образом, механизмы *IPC* в *Windows* обеспечивают универсальные средства для разработки гибких и масштабируемых приложений, способных обмениваться данными и взаимодействовать как в пределах одной системы, так и в распределенной среде. Их использование способствует повышению производительности, улучшению управления ресурсами и созданию сложных, взаимосвязанных программных решений.[2]

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

В данной программе реализуется функциональность сервера логирования и клиента, которые взаимодействуют через именованный канал. Основное назначение системы — сбор и протоколирование сообщений от нескольких процессов-клиентов с добавлением метаинформации. Программа включает следующие ключевые функции сервера:

*«void log\_message(const char\* client\_id, const char\* message)»*. Выполняет запись полученного сообщения в файл лога:

– Получает идентификатор клиента и его сообщение;

– Формирует строку записи, включая временную метку, идентификатор клиента и текст сообщения;

– Открывает (или создает) файл *server\_log.txt* в режиме добавления;

– Записывает данные в файл, используя синхронизацию с помощью мьютекса *logMutex*, чтобы избежать коллизий при одновременной записи от нескольких потоков.

*«DWORD WINAPI client\_handler(LPVOID param)»*. Обрабатывает подключение отдельного клиента в новом потоке:

– Читает данные из именованного канала, ассоциированного с клиентом;

– Для каждого сообщения вызывает функцию *log\_message* для записи данных в лог;

– Закрывает канал после завершения обработки.

*«int main()»*. Основная функция сервера:

– Создает файл *server\_log.txt* для хранения сообщений;

– Инициализирует мьютекс *logMutex* для синхронизации потоков;

– Запускает цикл ожидания подключений клиентов через именованный канал *«\\\\.\\pipe\\LogPipe»*;

– При подключении клиента создает новый поток для его обработки, ограничивая общее количество потоков значением *MAX\_CLIENTS*;

– Завершает выполнение, закрывая все созданные дескрипторы и потоки.

Программа содержит следующие ключевые функции клиента:

*«int main()»*. Основная функция клиента:

– Подключается к именованному каналу *«\\\\.\\pipe\\LogPipe»*, повторяя попытки с интервалом в 1 секунду до успешного соединения;

– Запрашивает ввод сообщения у пользователя;

– Отправляет введенное сообщение на сервер с помощью функции *WriteFile*;

– Закрывает дескриптор канала при завершении работы.

При запуске сервера выводится сообщение об ожидании подключения клиента (рисунок 3.1).

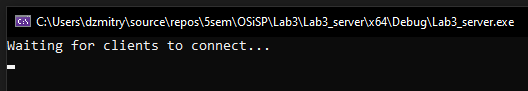


Рисунок 3.1 – Ожидание подключения к серверу

При запуске клиента, он подключается к каналу сервера, о чем сообщит сервер (рисунок 3.2).

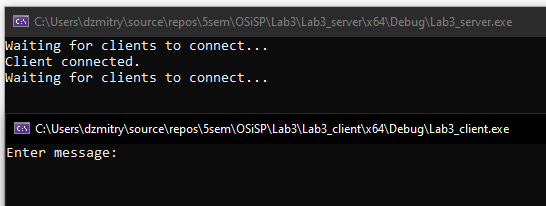


Рисунок 3.2 – Подключение клиента к серверу

Далее клиент может ввести сообщение, которое передастся по каналу на сервер и затем будет записано в файл. В файле хранятся записи о сообщениях, переданных клиентами. Запись содержит дату и время отправки сообщения, идентификатор клиента и само сообщение. Содержимое файла показано на рисунке 3.3.

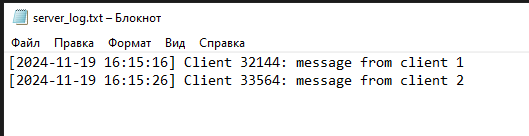


Рисунок 3.3 – Содержимое файла

Программа позволяет подключение сразу нескольких клиентов к одному серверу и передачу сообщений от этих клиентов с последующей записью данных сообщений в общий файл. Взаимодействие между процессами организовано посредством именованного канала, который создается на сервере и к которому подключаются клиенты.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы по реализации сервера логирования и клиента были изучены и применены механизмы межпроцессного взаимодействия (*IPC*) на основе именованных каналов. Полученные результаты подчеркивают важность использования эффективных методов организации обмена данными между процессами в условиях многопользовательской среды.

Программа продемонстрировала способность обрабатывать параллельные запросы от нескольких клиентов, обеспечивая корректность и последовательность записи данных в лог-файл. Использование мьютекса для синхронизации потоков позволило избежать коллизий при одновременной записи, что подтверждает необходимость применения механизмов синхронизации в многопоточных приложениях. Добавление временных меток и идентификаторов источников к записям в лог-файле обеспечило структурированность данных, что делает систему удобной для анализа и отладки.

Тестирование показало, что именованные каналы являются удобным и надежным инструментом для организации передачи данных между процессами. Однако в условиях высокой нагрузки их производительность может быть ограничена за счет увеличения времени ожидания соединения или обработки запросов. Это подчеркивает важность выбора подходящих механизмов *IPC* в зависимости от требований приложения.

Система также доказала свою масштабируемость, позволяя одновременно обслуживать до десяти клиентов. В то же время наблюдались определенные ограничения, связанные с ограничением числа потоков и необходимостью управления ресурсами, что является важным аспектом при разработке многозадачных систем.

Таким образом, выполненная работа продемонстрировала преимущества и ограничения подходов к межпроцессному взаимодействию на основе именованных каналов, подчеркнув важность применения синхронизации и структурированного подхода к обработке данных. Полученные результаты могут быть полезны для разработки приложений, работающих в условиях многопользовательского доступа и требующих надежной обработки и хранения данных.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

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

[2] Межпроцессное взаимодействие (IPC) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/uwp/communication/interprocess-communication

# **ПРИЛОЖЕНИЕ А** **(обязательное)** **Исходный код программного продукта**

#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;

}