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**

Студент Царук В.А.

Преподаватель Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc177388377)

[2 Краткие теоритические сведения 4](#_Toc177388378)

[3 Описание функций программы 5](#_Toc177388379)

[3.1 Используемый IPC 5](#_Toc177388380)

3.2 Процесс сервер 5

3.3 Процесс клиент 6

[Список использованных источников 8](#_Toc177388383)

[Приложение А (обязательное) Исходный код программы 9](#_Toc177388384)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является разбор подходов, системных объектов и функций для обеспечения передачи данных между взаимодействующими процессами и/или совместной их обработки. Рассмотреть типичные проблемы, возникающие при организации взаимодействия, и пути их решения.

В качестве задачи необходимо реализовать приложение, демонстрирующее работу многозадачного комплекса с обменом (передачей) или совместным использованием данных несколькими процессами. Произвести анализ корректности (отсутствия коллизий). Оценить эффективность механизмов *IPC*.

Нужно реализовать многопользовательский (с мультиплексированием) вывод.

Есть процесс-сервер, который ожидает данные (сообщения) от нескольких источников (клиентов); запись сообщений в файл с соблюдением порядка поступления, снабжая дополнительной информацией (временная метка, идентификатор источника и т.п.) и с соблюдением определенногоформата.

Есть процессы-клиенты, которые передают сгенерированные сообщения серверу для их протоколирования.

В качестве *IPC* можно использовать именованные и неименованные каналы, почтовые ящики, очереди сообщений.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Процесс – это выполняющаяся программа. У каждого процесса существует его собственное адресное пространство-области памяти, к которым этот процесс имеет доступ. Процесс включает в себя один и более потоков управления, которые представляют собой последовательность исполняемых инструкций [1].

Межпроцессное взаимодействие – обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности *IPC*. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети [2].

Общая классификация *IPC*:

– сигнальные (*signal*): прерывания, сигналы, сообщения и т.п.;

– канальные (*pipe*): трубопроводы и сокеты различного вида;

– разделяемая память (*shared memory*);

– очереди сообщений (*message queue, MQ*) – обладают свойствами сообщений и каналов;

– средства синхронизации (*ISO*): семафоры, мьютексы, барьеры и т.п.;

– комбинированные (комплексные) решения.

Объекты *ISO* – *Interprocess Synchronization Objects*, разновидность *IPC*-объектов.

Общая идея: проверка и модификация некоторого признака (флага) перед доступом к критическому ресурсу. Значение флага отражает свободное или занятое состояние ресурса. Основная проблема: регулятор доступа к критическим ресурсам – тоже критический ресурс, обращения к нему – критическая секция. Рекурсивное замыкание требований.

Семафор – глобальная переменная-счетчик *S*, целочисленные неотрицательные значения, атомарно выполняемые примитивы для доступа *P(S)* и *V(S)*:

1 *P(S)* – условный декремент: если значение достигло 0, то ожидание ненулевого значения.

2 *V(S)* – безусловный инкремент счетчика.

Мьютекс – можно рассматривать как упрощенный двузначный семафор, состояние которого интерпретируют как «свободность» и «занятость», а примитивы доступа – «захват» и «освобождение». Попытка повторного захвата блокирует поток-инициатор до освобождения мьютекса другим потоком.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Для реализации многопользовательского (с мультиплексированием) вывода будем создавать несколько процессов-клиентов, которые будут передавать сгенерированные сообщения серверу для их протоколирования.

## **3.1 Используемый IPC**

В качестве интерфейса сервера будет использоваться именованный канал. Канал создается в функции main процесса сервера с помощью CreateNamedPipe. Задаются имя канала, буферы определенного размера для чтения и записи, максимальное количество клиентов и доступ к каналу – для чтения.

В процессе клиента идет подключение к именованному каналу с помощью CreateFile, в параметрах которого указывается имя канала, право записи в канал. OPEN\_EXISTING указывает, что клиент подключается к уже существующему каналу.

## **3.2 Процесс сервер**

Процесс сервер создает или открывает текстовый файл для логирования, затем закрывает. Создается мьютекс с помощью CreateMutex для синхронизации доступа к файлу. Далее идет создания непосредственно именованного канала с помощью CreateNamedPipe с доступом для чтения и максимальным количеством клиентов. ConnectNamedPipe устанавливает соединение с клиентом, а сервер выводит сообщение о подключении. Сервер создаёт поток client\_handler для каждого нового клиента, если не достигнут лимит MAX\_CLIENTS. Если лимит превышен, подключение отклоняется. Функция ReadFile читает данные из канала в buffer. Цикл продолжается, пока ReadFile успешно получает данные. После каждого сообщения вызывается log\_message, записывающая его в файл.

Результат работы сервера (рисунок 3.1).

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

Рисунок 3.1 – Результаты работы сервера

Сервер принимает сообщения от клиентов через именованный канал и записывает их в файл. При записи в файл указывается время записи и идентификатор клиента (рисунок 3.2).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3.2 – Cодержимое файла server\_log.txt

**3.3 Процесс клиент**

Процесс клиент подключается к именованному каналу, который создал процесс сервер. Указывается имя канала, GENERICWRITE для того, чтобы клиент имел право только на запись в канал. OPEN\_EXISTING указывает, что клиент подключается к уже существующему каналу. Считывается ввод пользователя в buffer определенного размера. Через буфер идет запись в канал, который указывается через дескриптор. Наконец, идет закрытие дескриптора канала. Результаты работы клиента (рисунок 3.3).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3.3 – Результаты работы клиента

Клиент имеет возможность отправки сообщений через именованный канал на клиент сервер. Таким образом, присутствует возможность запуска нескольких клиентов и, тем самым, происходит запись в файл через канал.

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы был реализован многопользовательский (с мультиплексированием) вывод. Реализованы процесс клиент и процесс сервер. Процесс клиент передает сгенерированное сообщение серверу для их протоколирования. Клиент подключается к каналу, через который передается сообщение. Процесс сервер создает непосредственно IPC объект. В качестве IPC объекта использовался именованный канал, что позволяет передавать данные между процессами.

В качестве результата получили корректную передачу данных через именованный канал, который создается процессом сервером. Наличие нескольких клиентов никак не влияет на корректность программы. Используется мьютекс в качестве объекта синхронизации, что позволяет корректный доступ к файлу для записи сообщений. Из-за отсутствия одновременного доступа к файлу имеются корректные данные в файле, в который идет запись. Для корректного функционирования программы при чтении с помощью ReadFile происходит блокировка потока, который ждет поступления данных в канал. Тем временем, сервер активен – создает новый канал для каждого нового подключения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Руководство по межпроцессному взаимодействию (*IPC*) в *Linux* [Электронный ресурс]. – Электронный ресурс. – Режим доступа: https://habr.com/ru/

[2] Межпроцессное взаимодействие [Электронный ресурс]. – Электронный ресурс. – Режим доступа: https://ru.wikipedia.org/wiki/

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

Листинг 1 – client.cpp

#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 2 – server.cpp

#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)) { //lock function - ReadFile

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, //for reading

PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, //wait client

MAX\_CLIENTS,

BUFFER\_SIZE, //for writing

BUFFER\_SIZE, //for reading

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");

//waiting client

BOOL connected = ConnectNamedPipe(pipe, NULL) ? TRUE : (GetLastError() == ERROR\_PIPE\_CONNECTED); //set connect with client

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;

}