**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Операционные системы»**

**Тема: Межпроцессорное взаимодействие**

|  |  |  |
| --- | --- | --- |
| Студен гр. 2307 |  | Трофимов К.М |
| Преподаватель |  | Тимофеев А.В |

Санкт-Петербург

2024

# ****Цель работы.****

Исследовать инструменты и механизмы взаимодействия процессов в Windows.

выводами по заданию.

# Задание 4.1. Реализация решения задачи о читателях-писателях.

## Указания к выполнению.

1. Выполнить решение задачи о читателях-писателях, для чего

необходимо разработать консольные приложения «Читатель» и

«Писатель»:

 одновременно запущенные экземпляры процессов-читателей и

процессов-писателей должны совместно работать с буферной

памятью в виде проецируемого файла:

o размер страницы буферной памяти равен размеру

физической страницы оперативной памяти;

o число страниц буферной памяти равно сумме цифр в номере

студенческого билета без учета первой цифры.

 страницы буферной памяти должны быть заблокированы в

оперативной памяти (функция VirtualLock);

 длительность выполнения процессами операций «чтения» и

«записи» задается случайным образом в диапазоне от 0,5 до 1,5 сек.;

 для синхронизации работы процессов необходимо использовать

объекты синхронизации типа «семафор» и «мьютекс»;

 процессы-читатели и процессы-писатели ведут свои журнальные

файлы, в которые регистрируют переходы из одного «состояния» в

другое (начало ожидания, запись или чтение, переход к

освобождению) с указанием кода времени (функция TimeGetTime).

Для состояний «запись» и «чтение» необходимо также

запротоколировать номер рабочей страницы.

2. Запустите приложения читателей и писателей, суммарное

количество одновременно работающих читателей и писателей должно

быть не менее числа страниц буферной памяти. Проверьте

функционирование приложений, проанализируйте журнальные файлы

процессов, постройте сводные графики смены «состояний» для не менее

5 процессов-читателей и 5 процессов-писателей, дайте свои

комментарии относительно переходов процессов из одного состояния в

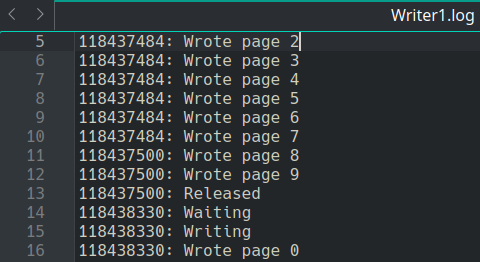
другое. Постройте графики занятости страниц буферной памяти

(проецируемого файла) во времени, дайте свои комментарии.

3. Подготовьте итоговый отчет с развернутыми выводами по

заданию.

## Вид Log-файлов



**Выполнение работы**

Для построения графиков из логов было написано несколько скриптов на python.

Первый скрипт преобразовывал данные из текстового формата в csv вида *Who, Time, Action*, где *Who* – *ReaderX* или *WriterX*, *Time* – количество тиков с момента запуска программы, *Action* – тип действий (*Waiting for lock*, *Reading/Writing*, *Lock Released*).

## Скрипт для конвертации логов в csv

import csv  
  
def extract\_data(log\_file, output\_csv):  
 # Initialize the list to hold extracted data  
 data = []  
 who = ""  
  
 # Open the log file for reading  
 with open(log\_file, 'r') as file:  
 lines = file.readlines()  
   
 # Process each line in the log file  
 for line in lines:  
 # Strip leading/trailing whitespace from the line  
 line = line.strip()  
 if not line:  
 continue  
   
 # Split the line into time and message  
 splitted = line.split(": ", 1)  
 if len(splitted)==1:  
 who = splitted[0]  
 print(who)  
 continue  
  
 time, message = splitted[0], splitted[1]  
  
 # Check if the line contains a "Waiting to" or "Writing/Reading" message  
 if "Waiting" in message or "Writing" in message or "Reading" in message:  
 # Determine if the activity is active or not  
 is\_active = "Active" if "Waiting" not in message else "Inactive"  
  
 # Subtract 8120000 from the time  
 time = int(time) - (8120000+1950 -1669)  
  
 # Append the extracted information as a tuple  
 data.append((who, time, is\_active))  
  
 # Sort the data based on the time  
 data.sort(key=lambda x: x[1])  
  
 # Write the extracted data to a CSV file  
 with open(output\_csv, 'w', newline='') as csvfile:  
 csvwriter = csv.writer(csvfile)  
 # Write the header row  
 csvwriter.writerow(["Who", "Time", "IsActive"])  
 # Write the data rows  
 csvwriter.writerows(data)  
  
  
  
# Specify the input log file and output CSV file  
log\_file = 'log.txt'  
output\_csv = 'output.csv'  
  
# Extract data and write to CSV  
extract\_data(log\_file, output\_csv)

**Скрипт для построения графиков**

import pandas as pdimport matplotlib.pyplot as plt# Read the CSV file into a DataFramedf = pd.read\_csv('output.csv')# select time intervaldf = df.loc[(df['Time']>80000) &( df['Time']<85000)]# Select rows you needdf = df[~df['Who'].str.contains('Reader')]# Get the unique names from the 'Who' columnunique\_names = df['Who'].unique()# Create a single plot for all namesplt.figure(figsize=(14, 8))# Plot each unique name on the same plotfor name in unique\_names: # Filter the DataFrame for the current name subset = df[df['Who'] == name] # Plot the data plt.plot(subset['Time'], subset['Action'], marker='o', linestyle='-', label=name)# Add plot detailsplt.title('Activity Status Over Time')plt.xlabel('Time')plt.ylabel('Activity')plt.yticks([0, 1, 2], ['Waiting', 'Working', 'Released'])plt.legend(title='Who')plt.grid(True)# Show the plotplt.show()

## Графики зависимости состояния от времени

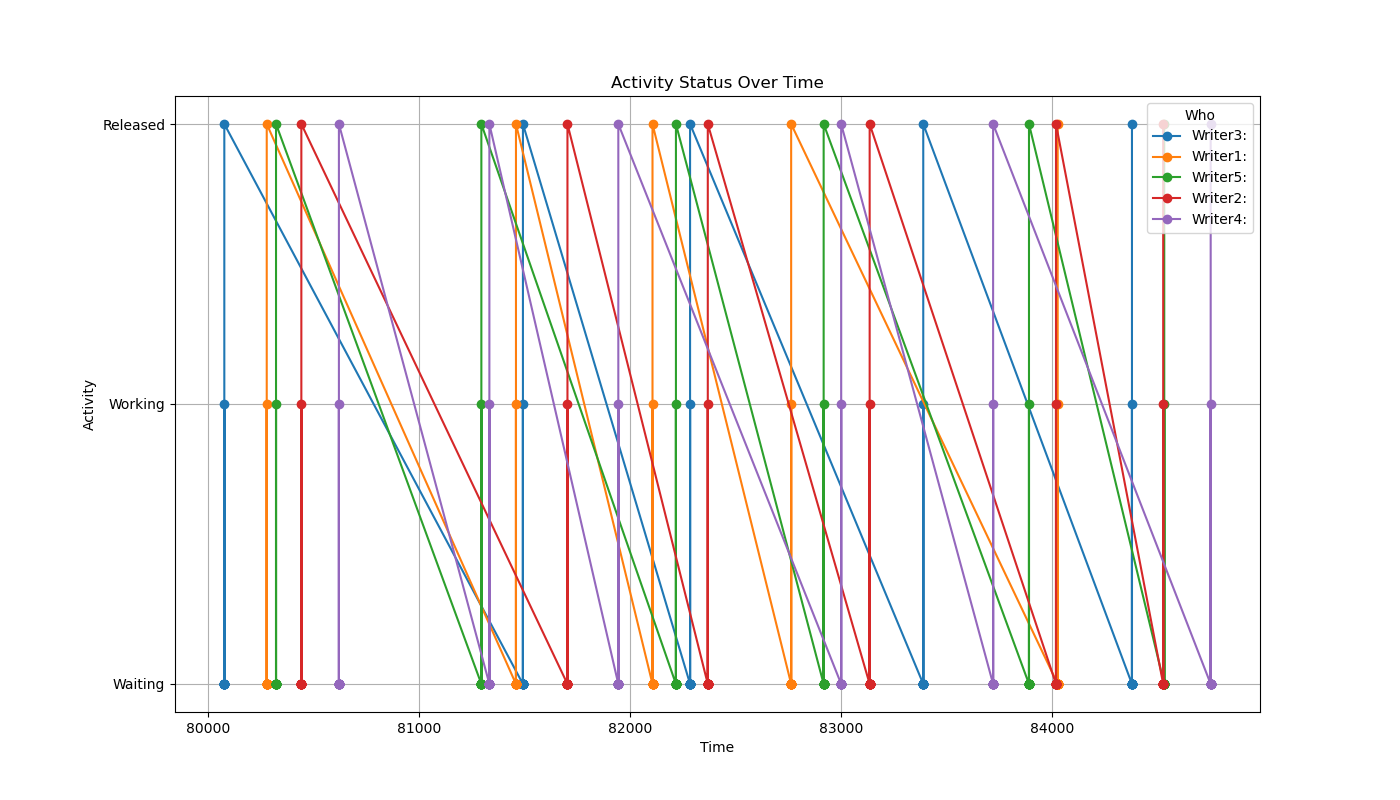


Рисунок 1 писатели

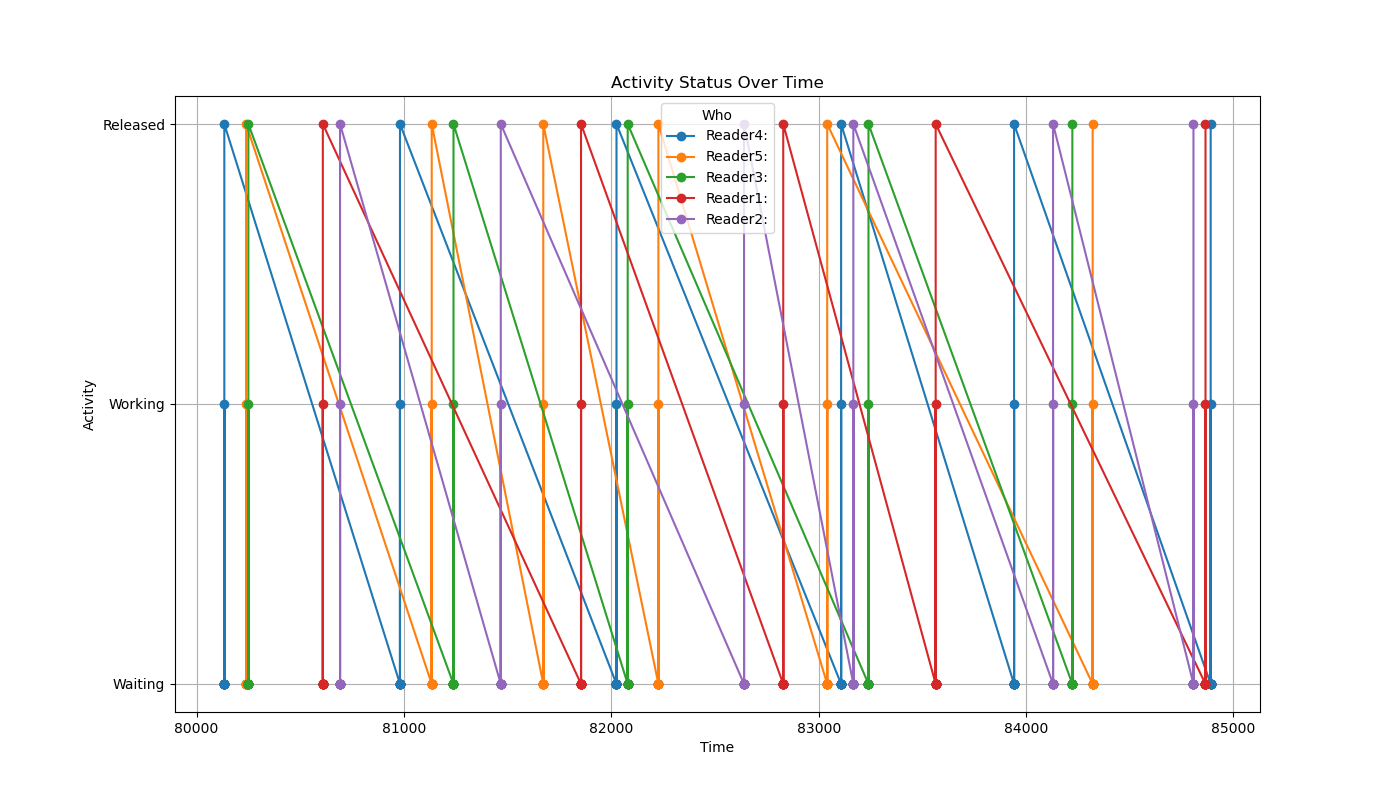
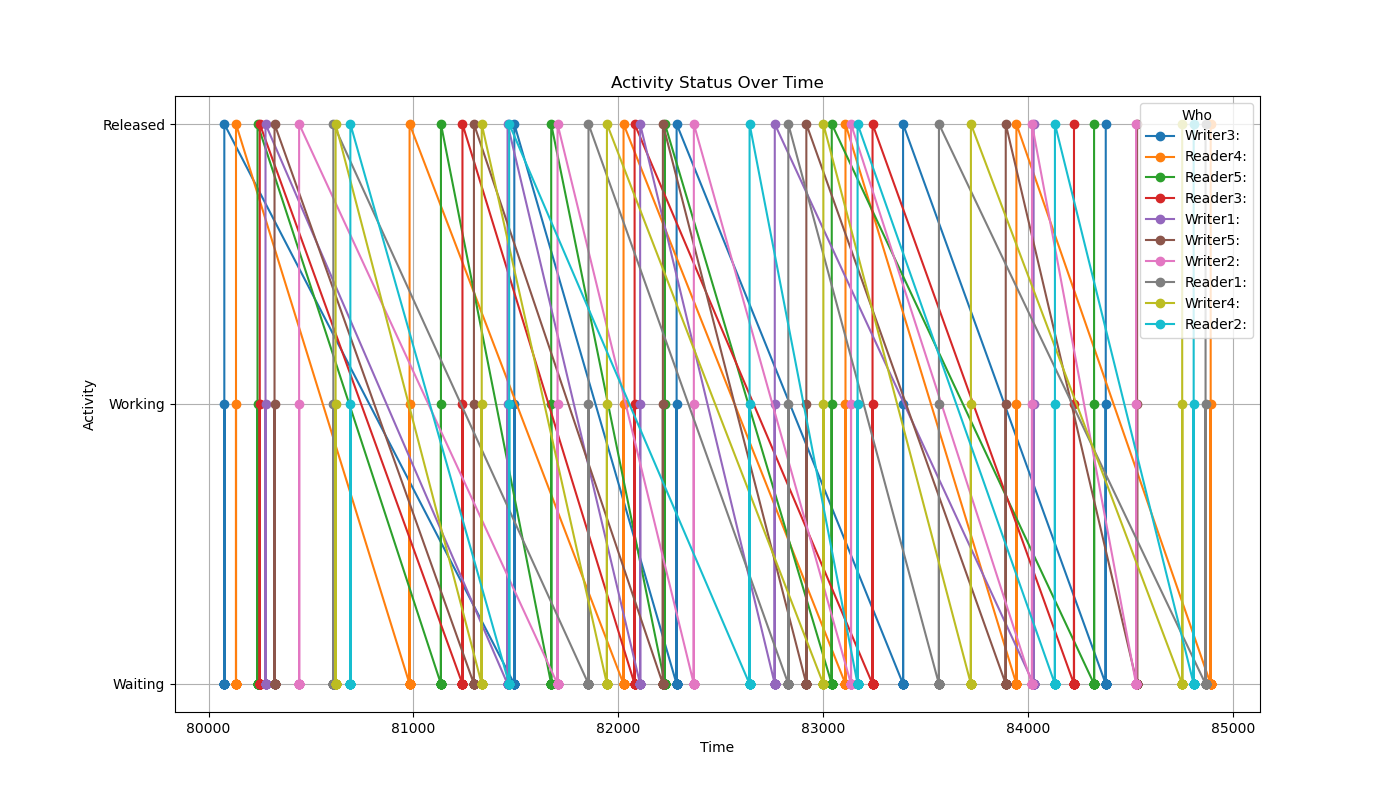


Рисунок 2 читатели

Рисунок 3 читатели и писатели

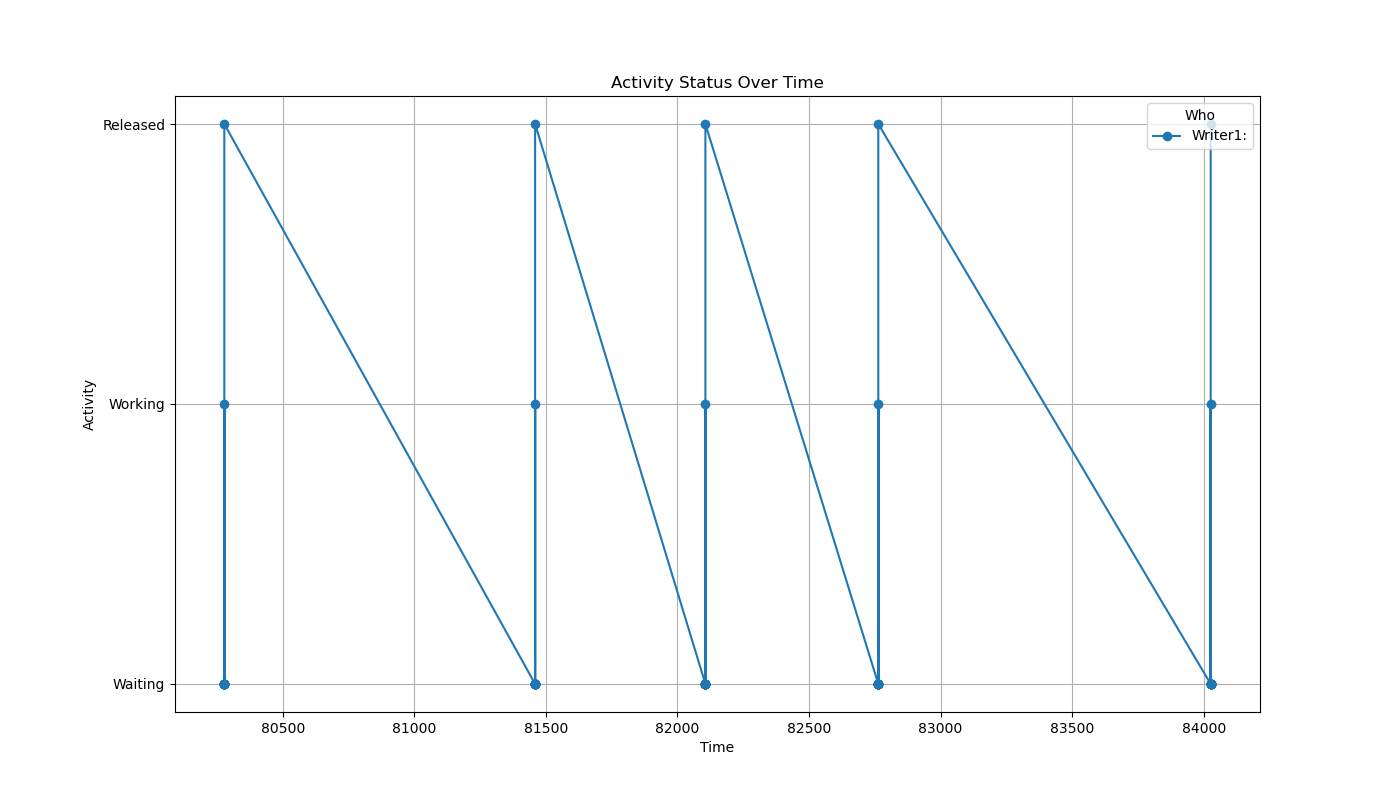


Рисунок 4 писатель 1

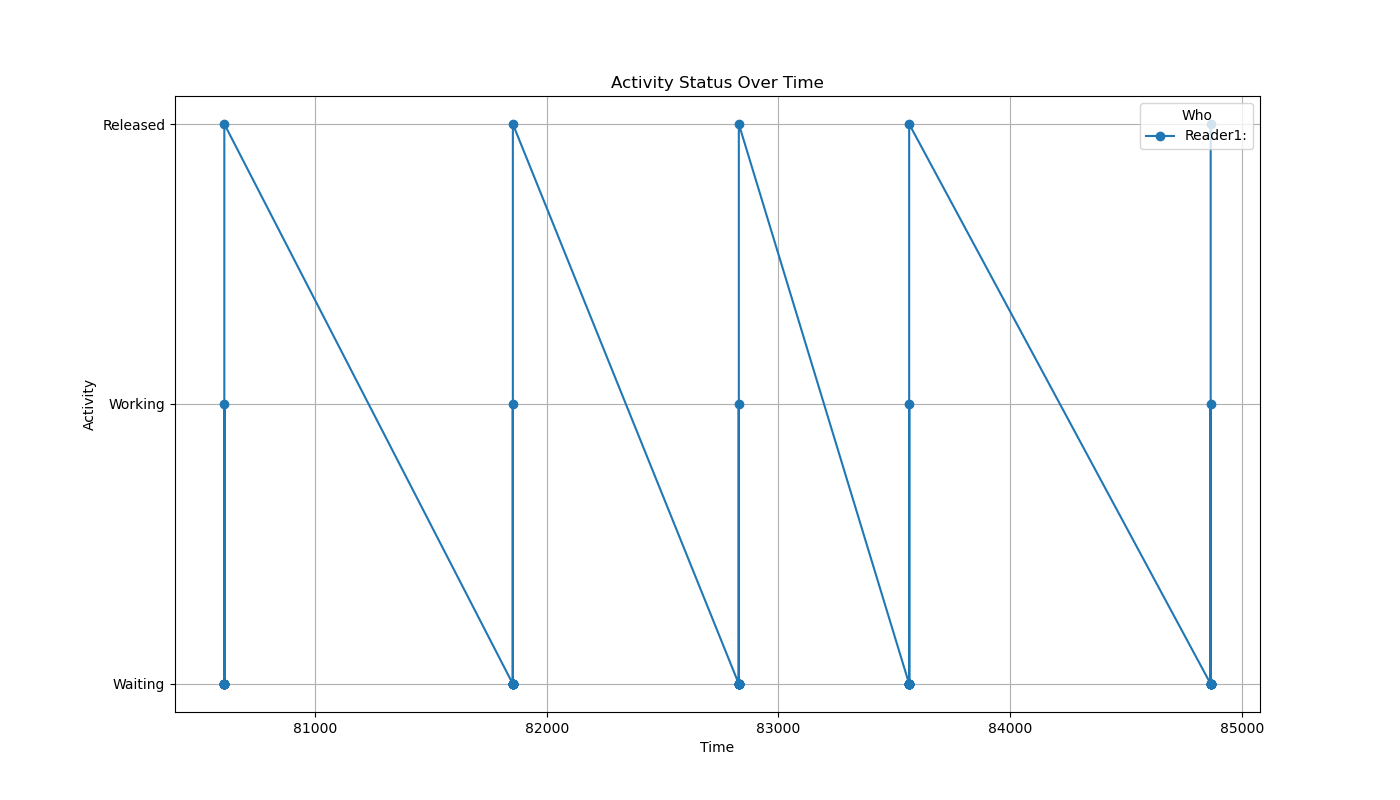


Рисунок 5 читатель 1

## Код программы

### Writer

// Writer.cpp  
  
#include <fstream>  
#include <iostream>  
#include <string>  
#include <windows.h>  
  
const int PAGE\_SIZE = 4096; // Размер страницы в байтах (обычно 4KB)  
const int STUDENT\_NUMBER\_SUM = 10;// Cумма цифр студака  
const int NUM\_PAGES = STUDENT\_NUMBER\_SUM;  
  
const char \*FILE\_MAPPING\_NAME = "SharedMemoryFile.txt";  
const char \*SEMAPHORE\_NAME = "BufferSemaphore";  
  
  
void LogEvent(const std::string &filename, const std::string &event) {  
 std::ofstream logFile(filename, std::ios::app);  
 if (logFile.is\_open()) {  
 DWORD time = timeGetTime();  
 logFile << time << ": " << event << std::endl;  
 logFile.close();  
 }  
}  
  
  
std::string GetLastErrorAsString() {  
 //Get the error message ID, if any.  
 DWORD errorMessageID = ::GetLastError();  
 if (errorMessageID == 0) {  
 return std::string();//No error message has been recorded  
 }  
  
 LPSTR messageBuffer = nullptr;  
  
 //Ask Win32 to give us the string version of that message ID.  
 //The parameters we pass in, tell Win32 to create the buffer that holds the message for us (because we don't yet know how long the message string will be).  
 size\_t size = FormatMessageA(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM | FORMAT\_MESSAGE\_IGNORE\_INSERTS,  
 NULL, errorMessageID, MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), (LPSTR) &messageBuffer, 0, NULL);  
  
 //Copy the error message into a std::string.  
 std::string message(messageBuffer, size);  
  
 //Free the Win32's string's buffer.  
 LocalFree(messageBuffer);  
  
 return message;  
}  
  
  
void Writer(int writerId) {  
 std::string logFilename = "Writer" + std::to\_string(writerId) + ".log";  
  
 // Открытие семафора и мьютекса  
 HANDLE hSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, SEMAPHORE\_NAME);  
  
 // Открытие проецируемого файла  
 HANDLE hMapFile = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, FALSE, NUM\_PAGES \* PAGE\_SIZE, FILE\_MAPPING\_NAME);  
 if (hMapFile == NULL) {  
 std::cout << "Could not create map view: "  
 << GetLastErrorAsString() << std::endl;  
  
  
 CloseHandle(hMapFile);  
  
 std::cin.get();  
 exit(1);  
 }  
 LPVOID pBuf = MapViewOfFile(hMapFile, FILE\_MAP\_WRITE, 0, 0, NUM\_PAGES \* PAGE\_SIZE);  
 if (pBuf == NULL) {  
 std::cout << "Could not map view of file: "  
 << GetLastErrorAsString() << std::endl;  
   
  
 CloseHandle(hMapFile);  
  
 std::cin.get();  
 exit(1);  
 }  
  
 while (true) {  
 LogEvent(logFilename, "Waiting");  
  
 // Ожидание семафора  
 WaitForSingleObject(hSemaphore, INFINITE);  
  
 LogEvent(logFilename, "Writing");  
  
 // Запись данных (пример)  
 for (int i = 0; i < NUM\_PAGES; i++) {  
 char\* page = (char\*)pBuf + i \* PAGE\_SIZE;  
 // Пример записи: установка первого байта страницы в значение идентификатора писателя  
 page[0] = 'A' + writerId;  
 LogEvent(logFilename, "Wrote page " + std::to\_string(i));  
 }  
  
  
  
 // Освобождение семафора  
 ReleaseSemaphore(hSemaphore, 1, NULL);  
 LogEvent(logFilename, "Released");  
  
 // Задержка  
 Sleep(500 + rand() % 1001); // От 0.5 до 1.5 сек.  
 }  
  
 UnmapViewOfFile(pBuf);  
 CloseHandle(hMapFile);  
 CloseHandle(hSemaphore);  
}  
  
int main(int argc, char\* argv[]) {  
 if (argc!= 2) {  
 std::cout << "Usage: " << argv[0] << " <writerId>" << std::endl;  
 return 1;  
 }  
 setlocale(LC\_ALL, "Russian");  
 srand(GetTickCount());  
 int writerId = std::atoi(argv[1]); // Уникальный идентификатор писателя  
 Writer(writerId);  
 return 0;  
}

### Reader

// Reader.cpp  
  
#include <fstream>  
#include <iostream>  
#include <string>  
#include <windows.h>  
  
const int PAGE\_SIZE = 4096; // Размер страницы в байтах (обычно 4KB)  
const int STUDENT\_NUMBER\_SUM = 10;// Cумма цифр студака  
const int NUM\_PAGES = STUDENT\_NUMBER\_SUM;  
  
const char \*FILE\_MAPPING\_NAME = "SharedMemoryFile.txt";  
const char \*SEMAPHORE\_NAME = "BufferSemaphore";  
  
  
void LogEvent(const std::string &filename, const std::string &event) {  
 std::ofstream logFile(filename, std::ios::app);  
 if (logFile.is\_open()) {  
 DWORD time = timeGetTime();  
 logFile << time << ": " << event << std::endl;  
 logFile.close();  
 }  
}  
  
  
void Reader(int readerId) {  
 std::string logFilename = "Reader" + std::to\_string(readerId) + ".log";  
  
 // Открытие семафора и мьютекса  
 HANDLE hSemaphore = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, SEMAPHORE\_NAME);  
  
 // Открытие проецируемого файла  
 HANDLE hMapFile = OpenFileMapping(FILE\_MAP\_READ, FALSE, FILE\_MAPPING\_NAME);  
 LPVOID pBuf = MapViewOfFile(hMapFile, FILE\_MAP\_READ, 0, 0, NUM\_PAGES \* PAGE\_SIZE);  
  
 while (true) {  
 LogEvent(logFilename, "Waiting");  
  
 // Ожидание семафора  
 WaitForSingleObject(hSemaphore, INFINITE);  
  
 LogEvent(logFilename, "Reading");  
  
 // Чтение данных (пример)  
 for (int i = 0; i < NUM\_PAGES; i++) {  
 char\* page = (char\*)pBuf + i \* PAGE\_SIZE;  
 // Пример чтения: вывод первого байта страницы  
 std::cout << "Reader " << readerId << " read page " << i << ": " << page[0] << std::endl;  
 LogEvent(logFilename, "Read page " + std::to\_string(i));  
 }  
  
  
 // Освобождение семафора  
 ReleaseSemaphore(hSemaphore, 1, NULL);  
 LogEvent(logFilename, "Released");  
 // Задержка  
 Sleep(500 + rand() % 1001); // От 0.5 до 1.5 сек.  
 }  
  
 UnmapViewOfFile(pBuf);  
 CloseHandle(hMapFile);  
 CloseHandle(hSemaphore);  
}  
  
int main(int argc, char\* argv[]) {  
 if (argc!= 2) {  
 std::cout << "Usage: " << argv[0] << " <readerId>" << std::endl;  
 return 1;  
 }  
 srand(GetTickCount());  
 int readerId = std::atoi(argv[1]); // Уникальный идентификатор читателя  
 Reader(readerId);  
 return 0;  
}

## ****Выводы по первой части.****

В данной лабораторной работе были созданы и протестированы консольные приложения "Читатель" и "Писатель", которые взаимодействуют с буферной памятью, реализованной в виде проецируемого файла. Размер страницы буферной памяти был выбран равным размеру физической страницы оперативной памяти, а количество страниц определялось суммой цифр номера студенческого билета (за исключением первой цифры). Страницы буферной памяти фиксировались в оперативной памяти с помощью функции VirtualLock.

Операции чтения и записи выполнялись с длительностью, случайно выбранной в диапазоне от 0,5 до 1,5 секунд. Для синхронизации процессов использовались семафоры, которые контролировали доступ к буферной памяти, обеспечивая её целостность и согласованность. Процессы вели журнал изменения состояний, фиксируя переходы между состояниями, включая время перехода и номер страницы. Эти журналы позволяли проследить последовательность действий каждого процесса и выявить возможные проблемы синхронизации.

Такой подход к решению задачи о читателях-писателях обеспечивает корректное и эффективное совместное использование буферной памяти. Использование семафоров предотвращает конфликты доступа, гарантируя, что в каждый момент времени к буферной памяти может обращаться только один процесс для записи. Логи процессов и графики состояний позволяют анализировать поведение процессов и выявлять проблемы, связанные с синхронизацией, что способствует повышению надежности и производительности системы.

**Задание 4.2. Использование именованных каналов для реализации сетевого межпроцессного взаимодействия.**

**Указания к выполнению.**

1. Создайте два консольных приложения с меню (каждая

выполняемая функция и/или операция должна быть доступна по

отдельному пункту меню), которые выполняют:

 приложение-сервер создает именованный канал (функция Win32

API – CreateNamedPipe), выполняет установление и отключение

соединения (функции Win32 API – ConnectNamedPipe,

DisconnectNamedPipe), создает объект «событие» (функция Win32

API – CreateEvent) осуществляет ввод данных с клавиатуры и их

асинхронную запись в именованный канал (функция Win32 API –

WriteFile), выполняет ожидание завершения операции ввода-

вывода (функция Win32 API – WaitForSingleObject);

 приложение-клиент подключается к именованному каналу

(функция Win32 API – CreateFile), в асинхронном режиме считывает

содержимое из именованного канала файла (функция Win32 API –

ReadFileEx) и отображает на экран.

2. Запустите приложения и проверьте обмен данных между

процессами. Запротоколируйте результаты в отчет. Дайте свои

комментарии в отчете относительно выполнения функций Win32 API.

3. Подготовьте итоговый отчет с развернутыми выводами по

заданию.

## Демонстрация работы программы

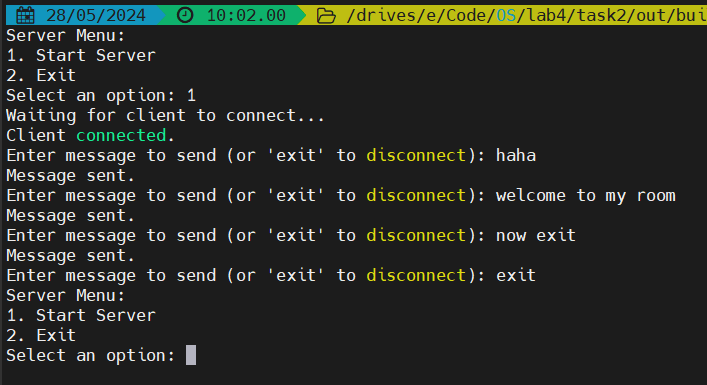


Рисунок 6 Отправка сообщения сервером

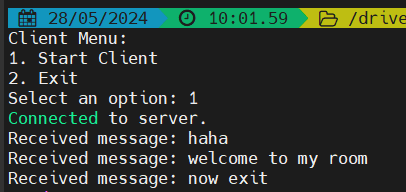


Рисунок 7 Получение сообщения клиентом

**Выполнение работы**

Сервер создает именованный канал с помощью функции CreateNamedPipe, Клиент подключается к именованному каналу с помощью функции CreateFile.

Для асинхронного ввода-вывода используются структуры OVERLAPPED. Сервер и клиент используют функции ReadFileEx и WriteFile с флагом FILE\_FLAG\_OVERLAPPED, что позволяет выполнять операции ввода-вывода без блокировки.

Сервер создает событие с помощью функции CreateEvent и использует его для синхронизации операций. Клиент обрабатывает завершение операций чтения с помощью функции обратного вызова ReadCompletionRoutine.

## Код программы

### Server

// Server.cpp  
  
#include <windows.h>  
#include <iostream>  
#include <string>  
#include <thread>  
  
const char \*PIPE\_NAME = R"(\\.\pipe\SamplePipe)";  
  
void Menu() {  
 std::cout << "Server Menu:\n";  
 std::cout << "1. Start Server\n";  
 std::cout << "2. Exit\n";  
 std::cout << "Select an option: ";  
}  
  
void StartServer() {  
 HANDLE hPipe;  
 HANDLE hEvent;  
 OVERLAPPED overlapped = {0};  
 char buffer[1024];  
 DWORD bytesWritten;  
  
 hPipe = CreateNamedPipe(  
 PIPE\_NAME,  
 PIPE\_ACCESS\_OUTBOUND | FILE\_FLAG\_OVERLAPPED,  
 PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT,  
 1,  
 1024,  
 1024,  
 0,  
 NULL  
 );  
  
 if (hPipe == INVALID\_HANDLE\_VALUE) {  
 std::cerr << "Failed to create named pipe. Error: " << GetLastError() << std::endl;  
 return;  
 }  
  
 hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);  
 if (hEvent == NULL) {  
 std::cerr << "Failed to create event. Error: " << GetLastError() << std::endl;  
 CloseHandle(hPipe);  
 return;  
 }  
  
 overlapped.hEvent = hEvent;  
  
 std::cout << "Waiting for client to connect..." << std::endl;  
 if (!ConnectNamedPipe(hPipe, &overlapped)) {  
 if (GetLastError() != ERROR\_IO\_PENDING) {  
 std::cerr << "Failed to connect named pipe. Error: " << GetLastError() << std::endl;  
 CloseHandle(hPipe);  
 CloseHandle(hEvent);  
 return;  
 }  
 }  
  
 WaitForSingleObject(hEvent, INFINITE);  
 std::cout << "Client connected." << std::endl;  
  
 while (true) {  
 std::cout << "Enter message to send (or 'exit' to disconnect): ";  
 std::string message;  
 std::getline(std::cin, message);  
  
 if (message == "exit") {  
 break;  
 }  
  
 ResetEvent(hEvent);  
 overlapped.hEvent = hEvent;  
  
 if (!WriteFile(hPipe, message.c\_str(), message.size() + 1, &bytesWritten, &overlapped)) {  
 if (GetLastError() != ERROR\_IO\_PENDING) {  
 std::cerr << "Failed to write to named pipe. Error: " << GetLastError() << std::endl;  
 break;  
 }  
 }  
  
 WaitForSingleObject(hEvent, INFINITE);  
 std::cout << "Message sent." << std::endl;  
 }  
  
 DisconnectNamedPipe(hPipe);  
 CloseHandle(hPipe);  
 CloseHandle(hEvent);  
}  
  
int main() {  
 int choice;  
 do {  
 Menu();  
 std::cin >> choice;  
 std::cin.ignore(); // Ignore the newline character left in the buffer  
 switch (choice) {  
 case 1:  
 StartServer();  
 break;  
 case 2:  
 std::cout << "Exiting..." << std::endl;  
 break;  
 default:  
 std::cout << "Invalid option. Try again." << std::endl;  
 break;  
 }  
 } while (choice != 2);  
  
 return 0;  
}

### Client

// Client.cpp  
#include <windows.h>  
#include <iostream>  
#include <string>  
  
const char \*PIPE\_NAME = R"(\\.\pipe\SamplePipe)";  
  
void Menu() {  
 std::cout << "Client Menu:\n";  
 std::cout << "1. Start Client\n";  
 std::cout << "2. Exit\n";  
 std::cout << "Select an option: ";  
}  
  
VOID CALLBACK  
ReadCompletionRoutine(DWORD  
dwErrorCode,  
DWORD dwNumberOfBytesTransfered, LPOVERLAPPED  
lpOverlapped) {  
 if (dwErrorCode == 0) {  
 std::cout << "Received message: " << (char\*)lpOverlapped->hEvent <<  
 std::endl;  
 } else {  
 std::cerr << "Read error: " << dwErrorCode <<  
 std::endl;  
 }  
}  
  
void StartClient() {  
 HANDLE hPipe;  
 OVERLAPPED overlapped = {0};  
 char buffer[1024];  
 DWORD bytesRead;  
  
 hPipe = CreateFile(  
 PIPE\_NAME,  
 GENERIC\_READ,  
 0,  
 NULL,  
 OPEN\_EXISTING,  
 FILE\_FLAG\_OVERLAPPED,  
 NULL  
 );  
  
 if (hPipe == INVALID\_HANDLE\_VALUE) {  
 std::cerr << "Failed to connect to named pipe. Error: " << GetLastError() << std::endl;  
 return;  
 }  
  
 std::cout << "Connected to server." << std::endl;  
  
 while (true) {  
 memset(buffer, 0, sizeof(buffer));  
 overlapped.hEvent = buffer;  
  
 if (!ReadFileEx(hPipe, buffer, sizeof(buffer), &overlapped, ReadCompletionRoutine)) {  
 if (GetLastError() != ERROR\_IO\_PENDING) {  
 std::cerr << "Failed to read from named pipe. Error: " << GetLastError() << std::endl;  
 break;  
 }  
 }  
  
 SleepEx(INFINITE, TRUE);  
 }  
  
 CloseHandle(hPipe);  
}  
  
int main() {  
 int choice;  
 do {  
 Menu();  
 std::cin >> choice;  
 std::cin.ignore(); // Ignore the newline character left in the buffer  
 switch (choice) {  
 case 1:  
 StartClient();  
 break;  
 case 2:  
 std::cout << "Exiting..." << std::endl;  
 break;  
 default:  
 std::cout << "Invalid option. Try again." << std::endl;  
 break;  
 }  
 } while (choice != 2);  
  
 return 0;  
}

## Выводы по второй части

## В данном проекте были разработаны и протестированы консольные приложения "Клиент" и "Сервер", которые взаимодействуют через именованный канал. Эти приложения позволяют устанавливать двустороннюю связь, при которой клиент может получать сообщения от сервера, а сервер - отправлять их.