Минобрнауки России  
Федеральное государственное автономное образовательное  
Учреждение высшего образования  
«Санкт-Петербургский государственный электротехнический  
Университет им. В.И. Ульянова (Ленина)»  
(СПГЭТУ «ЛЭТИ»)  
Факультет компьютерных технологий и информатики

Кафедра вычислительной техники  
  
  
  
**Отчет по лабораторной работе №4**

**на тему: «Межпроцессное взаимодействие»**

**по дисциплине «Операционные системы»**

Выполнил студент группы 9308: Семенов А.И.

Принял: к.т.н., доцент Тимофеев А.В.

Санкт-Петербург  
2021 г.

Содержание

[Цель работы 3](#_Toc89199665)

[Реализация многопоточного приложения с использованием функций Win32 API 3](#_Toc89199666)

[Указания к выполнению 3](#_Toc89199667)

[Результаты выполнения программы 4](#_Toc89199668)

[Вывод по заданию 8](#_Toc89199669)

[Реализация многопоточного приложения с использованием технологии OpenMP 9](#_Toc89199670)

[Указания к выполнению 9](#_Toc89199671)

[Результаты выполнения программы 10](#_Toc89199672)

[Вывод по заданию 11](#_Toc89199673)

[Вывод 12](#_Toc89199674)

# Цель работы

Исследовать инструменты и механизмы взаимодействия процессов в Windows.

## 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:
  + размер страницы буферной памяти равен размеру физической страницы оперативной памяти;
  + 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.

## Результаты выполнения программы

Задача о читателях-писателях реализована в виде трех программ:

* главная: создает проецируемый файл и объекты синхронизации, а также процессы читателей и писателей;
* читатель: считывает данные со страницы, в которую писатель что-либо записал;
* писатель: записывает данные в свободную страницу.

Первая программа для запуска принимает следующие параметры: число процессов и число работы этих процессов (сколько раз писатель должен записать данные и сколько раз читатель должен считать данные).

Пример запуска и сама работа главного приложения представлены на рисунке 1.



Рисунок – Запуск и работа главного приложения

Сами же процессы писателей и читателей имеют свои файлы протоколирования действия: writer.log – для писателей, reader.log – для читателей. Пример записанных данных в эти файлы для указанных выше параметрах запуска главного приложения представлены на рисунках 2 и 3 соответственно.

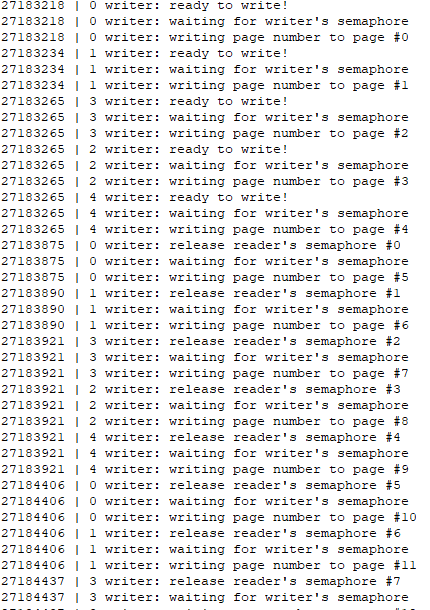


Рисунок – Данные в writer.log

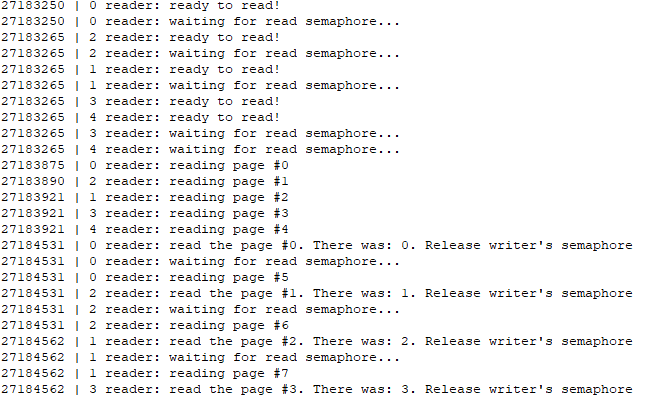


Рисунок – Данные в reader.log

Если отражать полученные данные на графике состояний каждого процесса в отдельности, то получим график, указанный на рисунке 4, где ниже синей линии – писатели, выше – читатели.

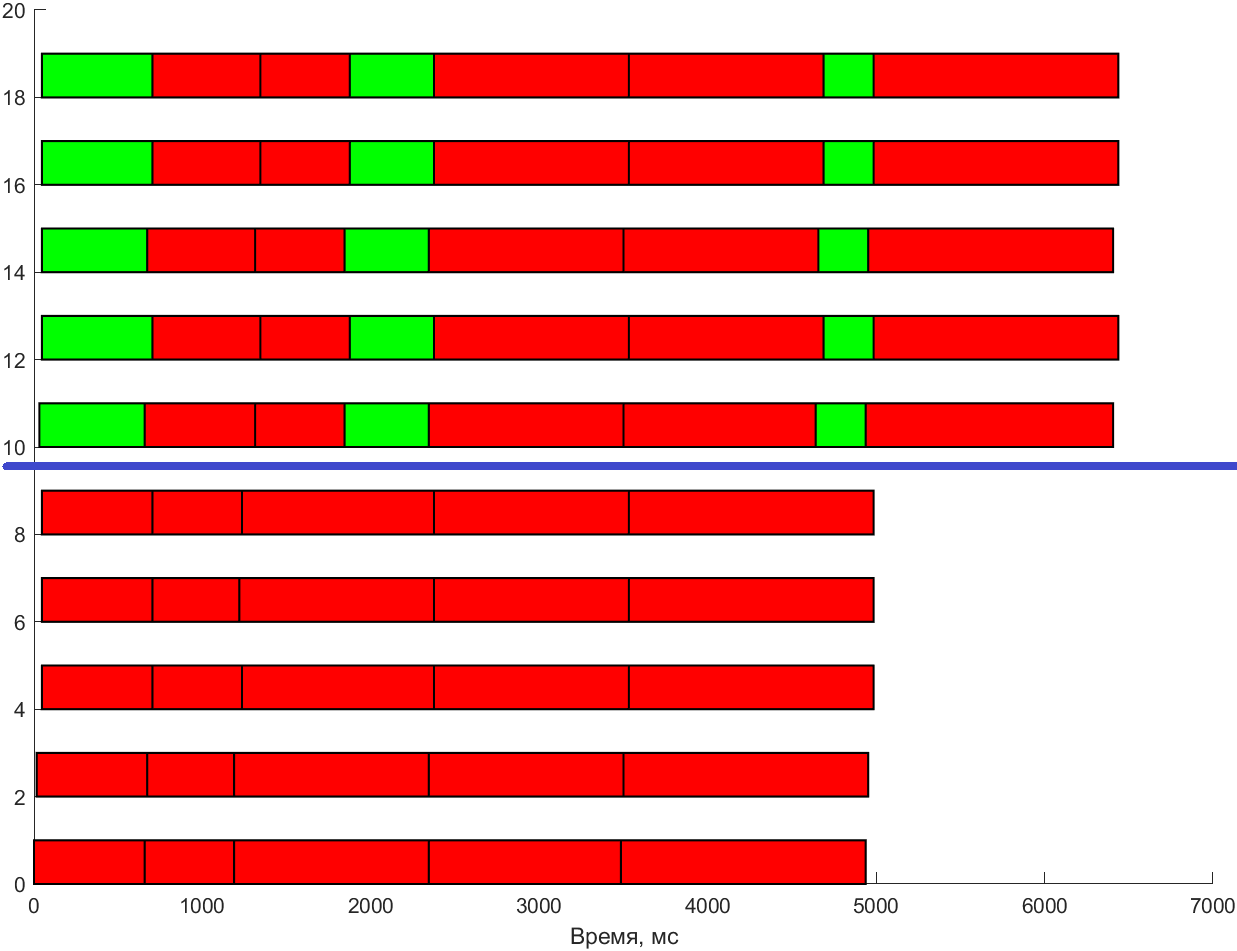


Рисунок – График состояний процессов

На этом графике отчетливо можно заметить, как читатели (они выше синей линии) в начале своего жизненого цикла ожидают, пока писатели что-то напишут в какую-либо страницу. Состояние ожидания отображается зеленым цветом. Сразу при записи одним из писателей каких-либо данных в страницу, читатель начинает процесс считывания данных. Состояние занятости – красный цвет.

Писатели в свою очередь в связи с большим числом страниц (20 против 5 писателей и 5 читателей) после записи данных в одну страницу сразу берутся за новую страницу и записывают туда данные (так как всегда остается не менее 10 свободных страниц, что будет показано дальше).

Все эти процессы записи/считывания проходят 5 раз, что соответствует количеству состояний занятости на графике.

Если же смотреть на занятость страниц в это время, то получим график, показанный на рисунке 5, где зеленый цвет отвечает за свободные страницы, синий – за действующих в данный момент писаталей, красный – за активных читателей.

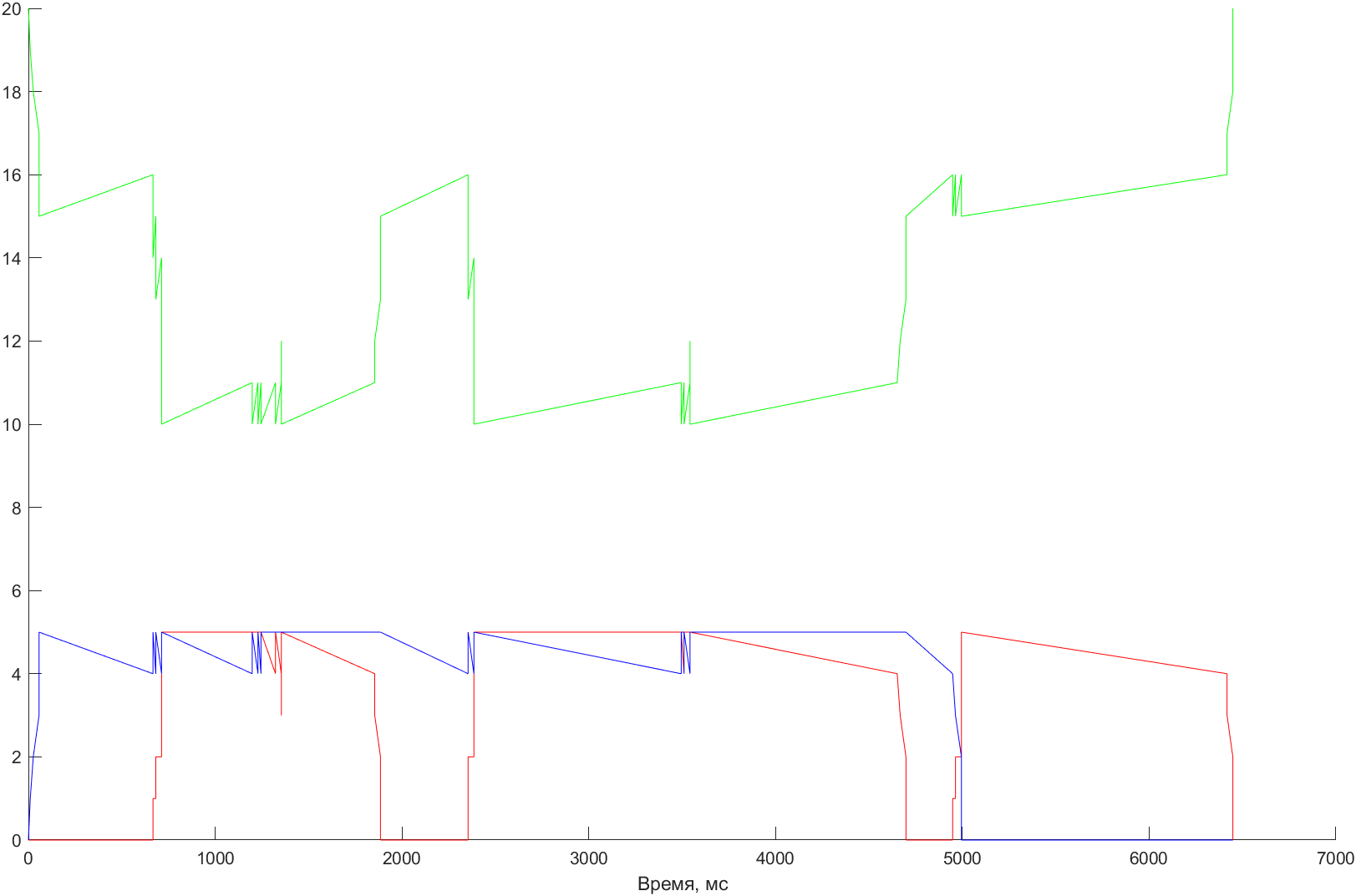


Рисунок – График занятости страниц

По графику можно заметить, что число свободных страниц никогда не опускается ниже 10, т.к. процессов, которые пишут в страницу/считывают с нее, в сумме дают ровно 10, а время их действий на данном промежутке времени не создает ситуации, когда писатели бы обгоняли читателей: успевали бы заполнять страницы наперед, пока читатели заняты считыванием данных с других страниц.

Для десяти процессов графики состояний процессов и занятости страниц показаны на рисунках ? и ? соответственно.

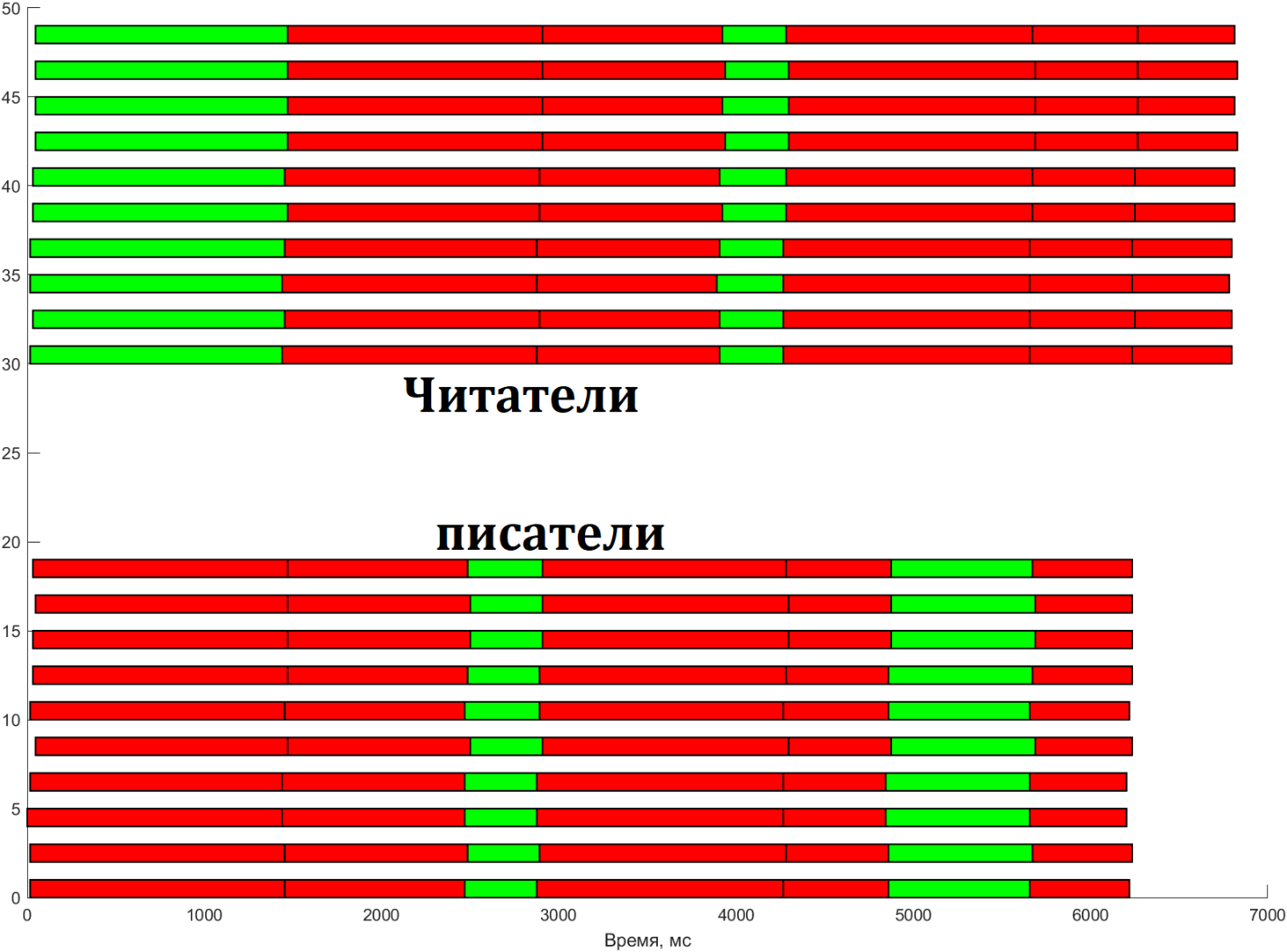


Рисунок – Состояния процессов (10 читателей/писателей)

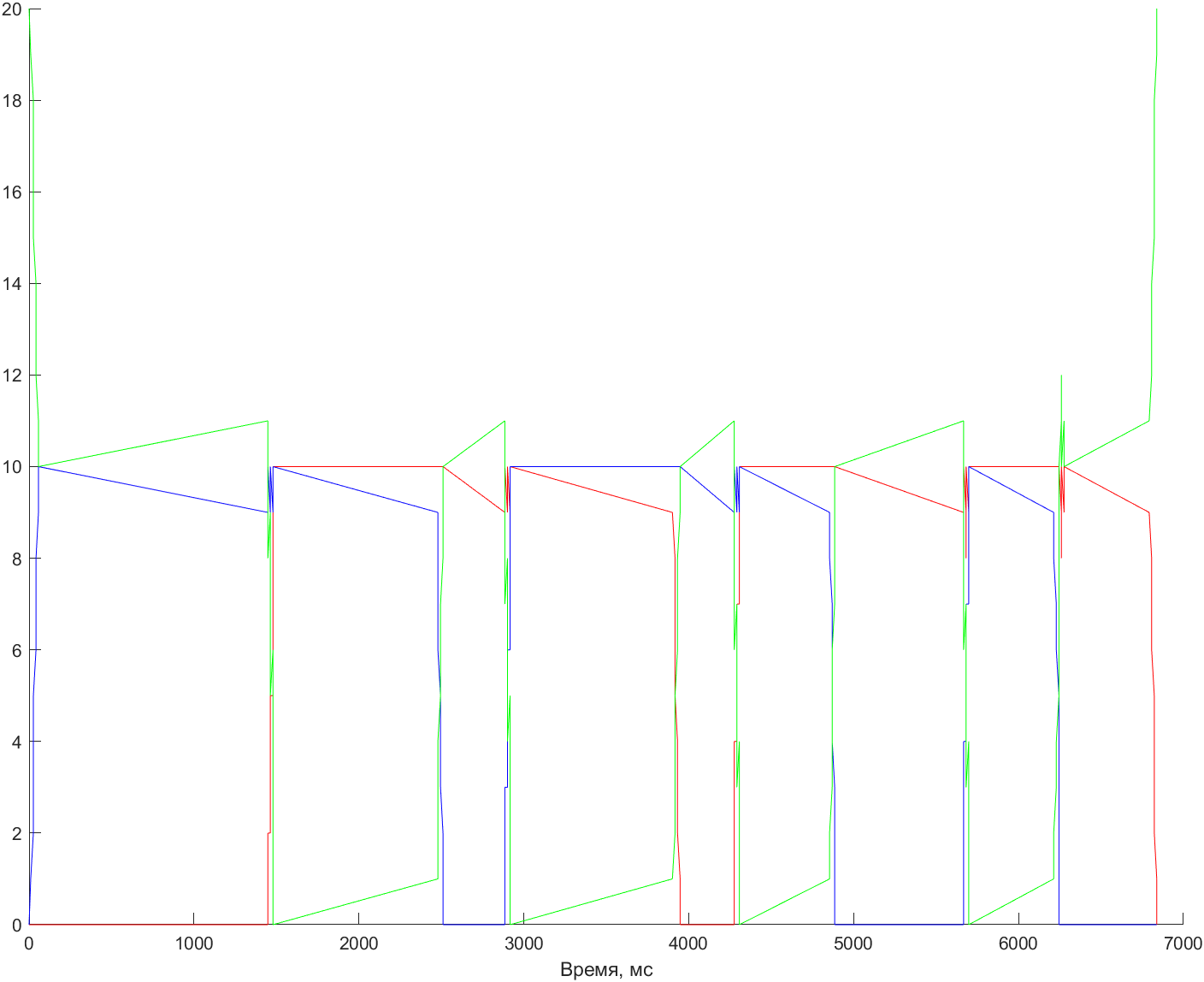


Рисунок – Занятость страниц (10 читателей/писателей)

По данным графикам уже можно заметить, что появляются моменты, когда и читатели, и писатели ожидают страницу для работы над ней. При сопоставлении графиков можно подтвердить моменты ожидания тем, что в это время нет свободных страниц (зеленый цвет).

Для 40 процессов (для 2 повторов чтения/записи одним процессом) графики уже будут иметь следующий вид:

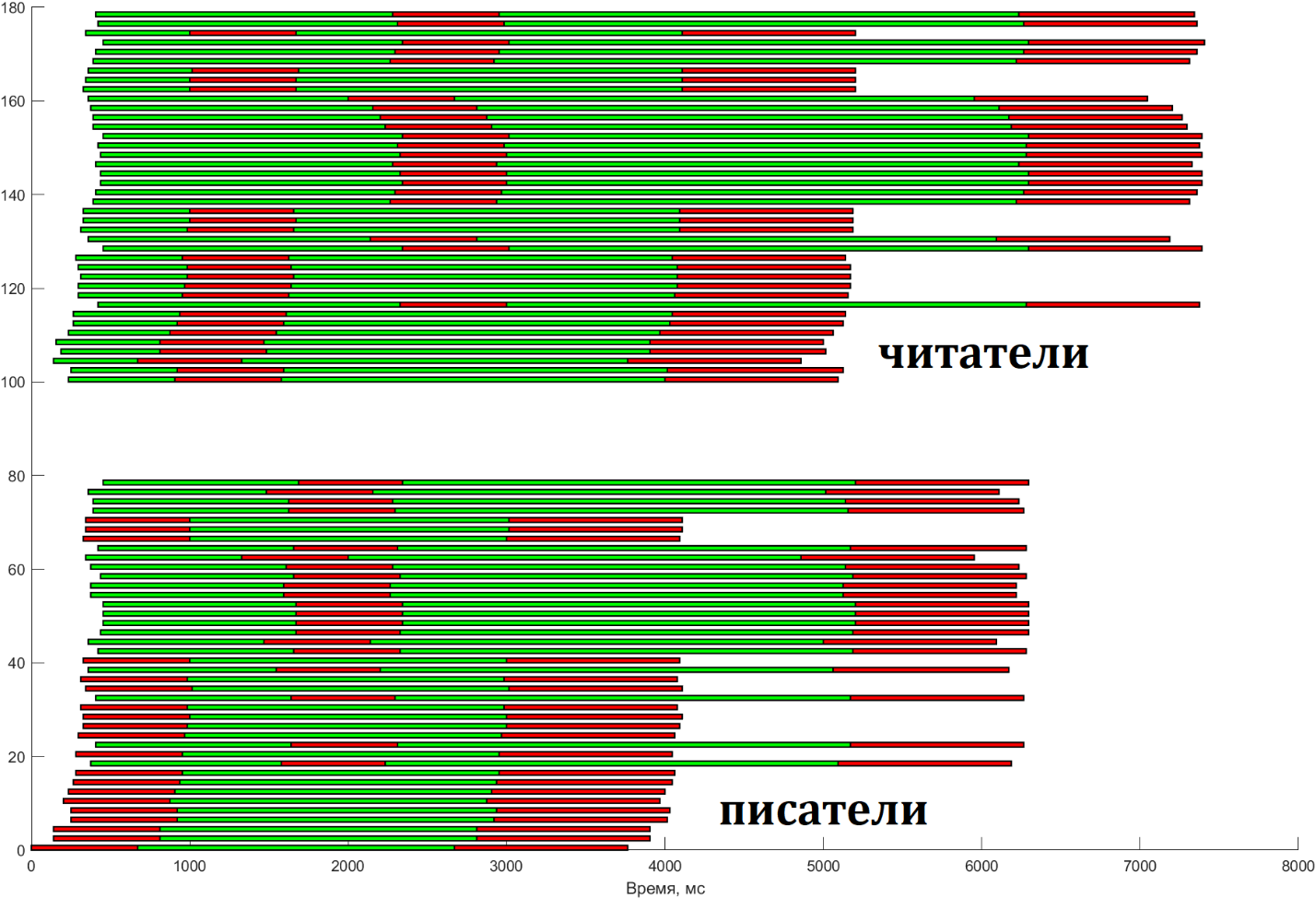


Рисунок – Состояния процессов (40 писателей/читателей)

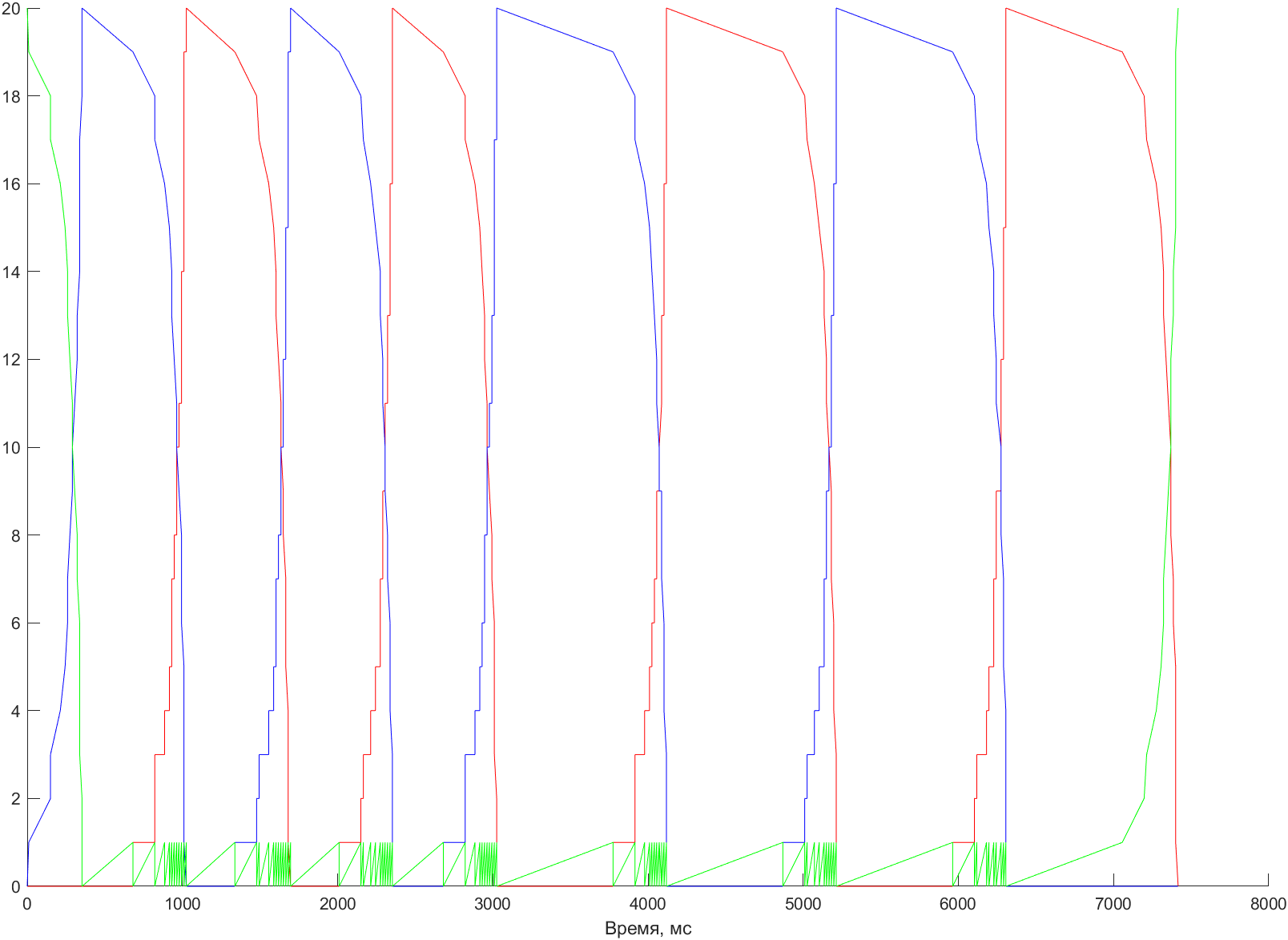


Рисунок – Занятость страниц (40 писателей/читателей)

## Вывод по заданию

Были изучены механизмы создания и управления процессами и потоками в ОС Windows. По результатам замера можно заметить, что оптимальное число потоков, работающих в процессе для вычисления числа Пи, равно числу потоков на самом процессоре. При увеличении числа потоков сверх тех, что есть у процессора время вычисления практически одинаковое, т.к. во время работы программы процессор мог бы занят другими задачами.

## 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 вводавывода (функция Win32 API – **WaitForSingleObject**);

2. Запустите приложения и проверьте обмен данных между процессами. Запротоколируйте результаты в отчет. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

## Результаты выполнения программы

Приложение сервер

Приложение “сервер” имеет меню, указанное на рисунке 1.

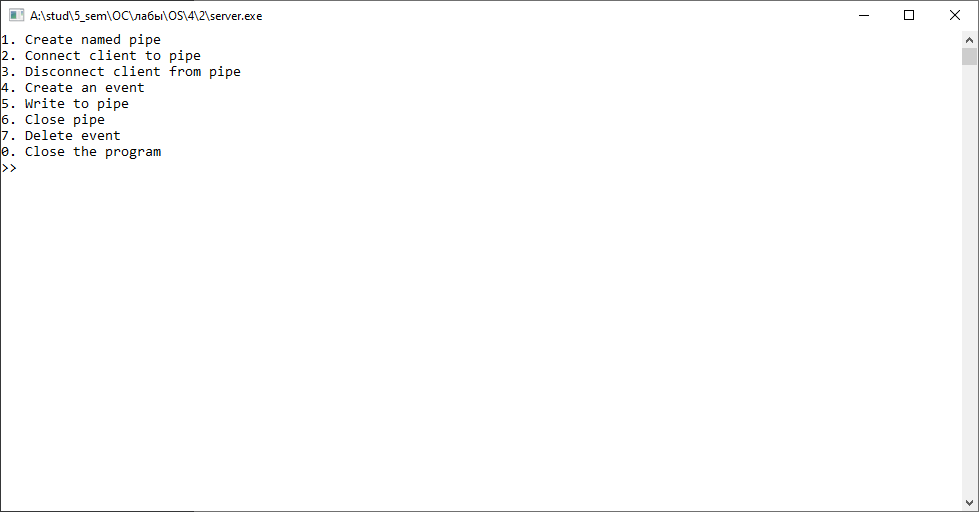


Рисунок – Меню приложения “сервер”

При выборе первого пункта пользователю предоставляется возможность назвать именнованный канал. Сам процесс создания канала показан на рисунке 2. Данное название понадобится потом в приложении “клиент”.

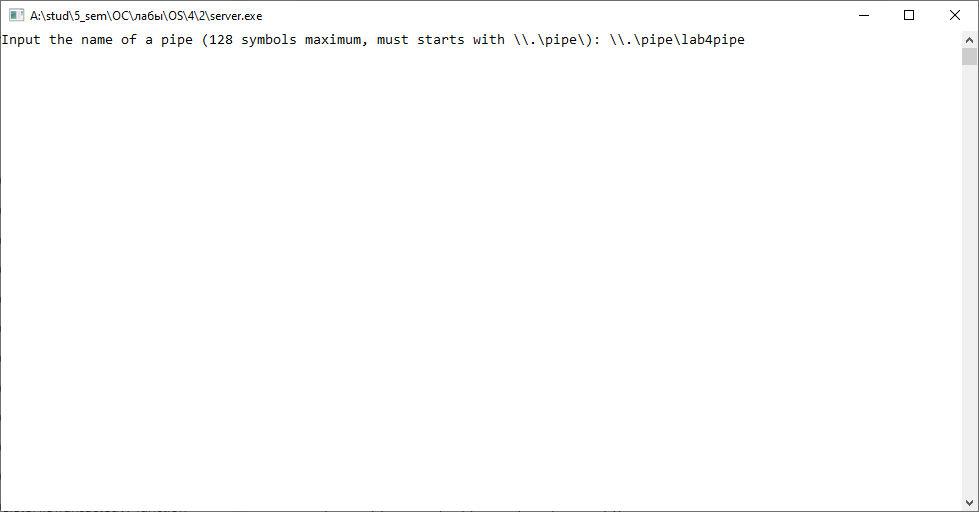


Рисунок – Создание именнованого канала

После создания канала стоит нажать пункт 2, чтобы подключить клиента к каналу. В этот момент в приложении “клиент”, меню которого представлено на рисунке 3 стоит выбрать пункт 1.

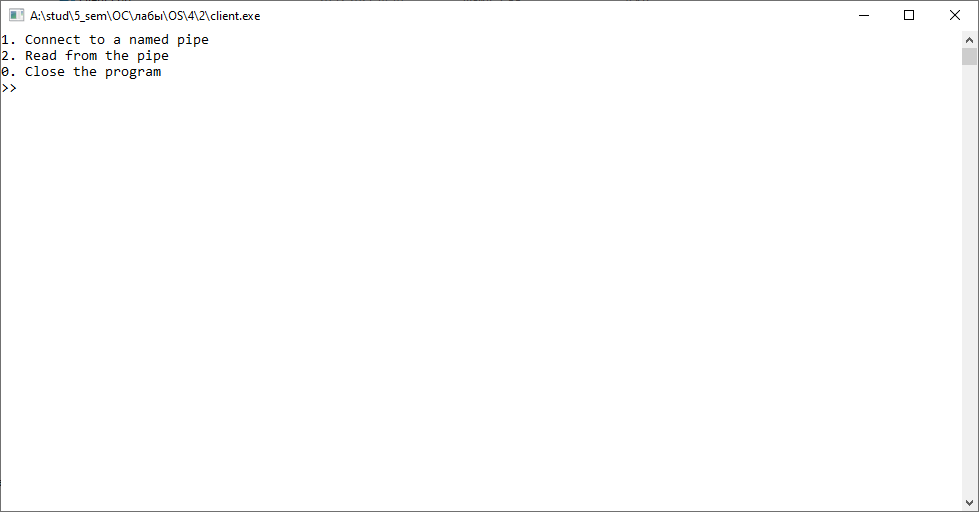


Рисунок – Меню приложения “клиент”

При успешном подключении на стороне сервера появится сообщение, показанное на рисунке 4, а у клиента – сообщение на рисунке 5.

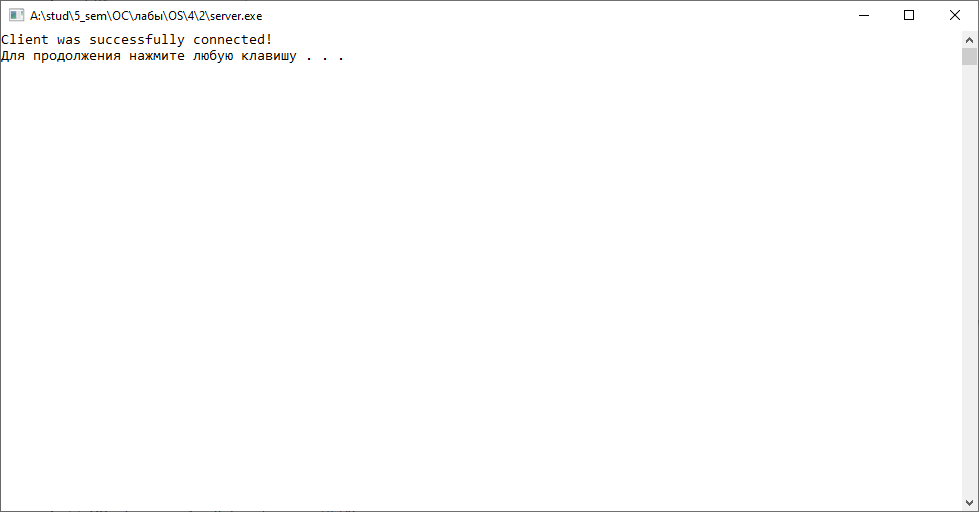


Рисунок – Сообщение об успешном подключении на стороне сервера

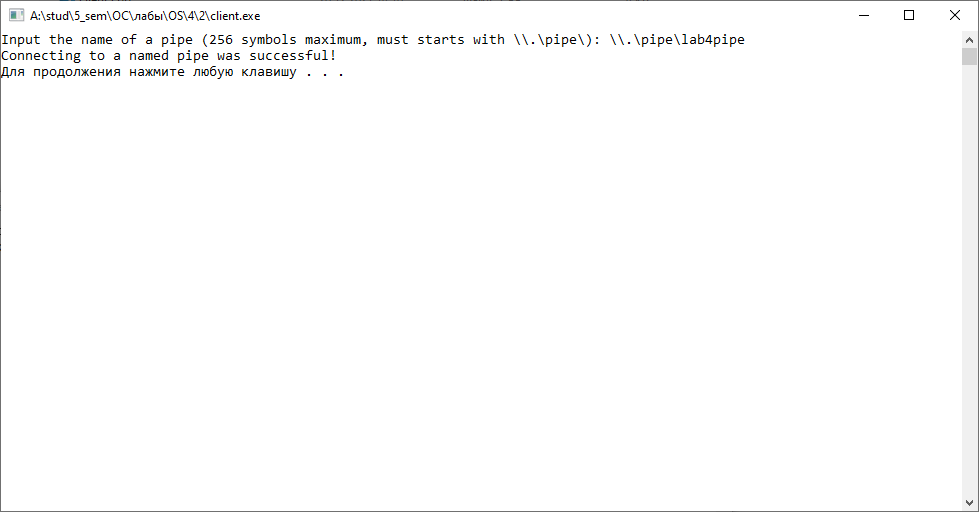


Рисунок – Сообщение об успешном подключении на стороне клиента

После данных действий серверу нужно создать событие, используемое при написании сообщения в канал. Создание события производится нажатием 4-го пункта меню. В случае успеха появится сообщение, показанное на рисунке 6.



Рисунок – Создание события

После создания события можно писать в канал, выбрав 5-й пункт меню (Write to pipe). На стороне клиента же нужно выбрать 2-й пункт меню (Read from the pipe). Демонстрация передачи сообщения представлена на рисунках 7 и 8 для сервера и клиента соответственно.



Рисунок – Написание сообщения в канал сервером



Рисунок – Получение сообщения клиентом

При ошибке создания канала или для пересоздания канала/события можно воспользоваться пунктами 6 и 7 на стороне сервера. При закрытии канала соединение с клиентом автоматически обрывается.

## Вывод по заданию

Были изучены функции Win32 API, позволяющие производить обмен между процессами с помощью именнованного канала.

Одно приложение в данном случае выступает в роли сервера, которое создает канал (функция CreateNamedPipe), ожидает подключения и подключает клиента к нему (функция ConnectNamedPipe) и передает ему сообщение через данный канал (функция WriteFile). Также сервер можно отключить клиента от канала (функция DisconnectNamedPipe).

Другое приложение (клиент) подключается к каналу (функция CreateFile) и считывает асинхронно сообщение из канала (ReadFileEx).

# Вывод

ы