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

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.

```
>a.exe 5 5
Everything's ok, press any to close the program
Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 – 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 параметров запуска главного приложения представлены на рисунках 2 и 3 соответственно.

```
27183218 | 0 writer: ready to write!
27183218 | 0 writer: waiting for writer's semaphore
27183218 | 0 writer: writing page number to page #0
27183234 | 1 writer: ready to write!
27183234 | 1 writer: waiting for writer's semaphore
27183234 | 1 writer: writing page number to page #1
27183265 | 3 writer: ready to write!
27183265 | 3 writer: waiting for writer's semaphore
27183265 | 3 writer: writing page number to page #2
27183265 | 2 writer: ready to write!
27183265 | 2 writer: waiting for writer's semaphore
27183265 | 2 writer: writing page number to page #3
27183265 | 4 writer: ready to write!
27183265 | 4 writer: waiting for writer's semaphore
27183265 | 4 writer: writing page number to page #4
27183875 | 0 writer: release reader's semaphore #0
27183875 | 0 writer: waiting for writer's semaphore
27183875 | 0 writer: writing page number to page #5
27183890 | 1 writer: release reader's semaphore #1
27183890 | 1 writer: waiting for writer's semaphore
27183890 | 1 writer: writing page number to page #6
27183921 | 3 writer: release reader's semaphore #2
27183921 | 3 writer: waiting for writer's semaphore
27183921 | 3 writer: writing page number to page #7
27183921 | 2 writer: release reader's semaphore #3
27183921 | 2 writer: waiting for writer's semaphore
27183921 | 2 writer: writing page number to page #8
27183921 | 4 writer: release reader's semaphore #4
27183921 | 4 writer: waiting for writer's semaphore
27183921 | 4 writer: writing page number to page #9
27184406 | 0 writer: release reader's semaphore #5
27184406 | 0 writer: waiting for writer's semaphore
27184406 | 0 writer: writing page number to page #10
27184406 | 1 writer: release reader's semaphore #6
27184406 | 1 writer: waiting for writer's semaphore
27184406 | 1 writer: writing page number to page #11
27184437 | 3 writer: release reader's semaphore #7
27184437 | 3 writer: waiting for writer's semaphore
```

Рисунок 2 – Данные в writer.log

```
27183250 | 0 reader: ready to read!
27183250 | 0 reader: waiting for read semaphore...
27183265 | 2 reader: ready to read!
27183265 | 2 reader: waiting for read semaphore...
27183265 | 1 reader: ready to read!
27183265 | 1 reader: waiting for read semaphore...
27183265 | 3 reader: ready to read!
27183265 | 4 reader: ready to read!
27183265 | 3 reader: waiting for read semaphore...
27183265 | 4 reader: waiting for read semaphore...
27183875 | 0 reader: reading page #0
27183890 | 2 reader: reading page #1
27183921 | 1 reader: reading page #2
27183921 | 3 reader: reading page #3
27183921 | 4 reader: reading page #4
27184531 | 0 reader: read the page #0. There was: 0. Release writer's semaphore
27184531 | 0 reader: waiting for read semaphore...
27184531 | 0 reader: reading page #5
27184531 | 2 reader: read the page #1. There was: 1. Release writer's semaphore
27184531 | 2 reader: waiting for read semaphore...
27184531 | 2 reader: reading page #6
27184562 | 1 reader: read the page #2. There was: 2. Release writer's semaphore
27184562 | 1 reader: waiting for read semaphore...
27184562 | 1 reader: reading page #7
27184562 | 3 reader: read the page #3. There was: 3. Release writer's semaphore
```

Рисунок 3 – 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.

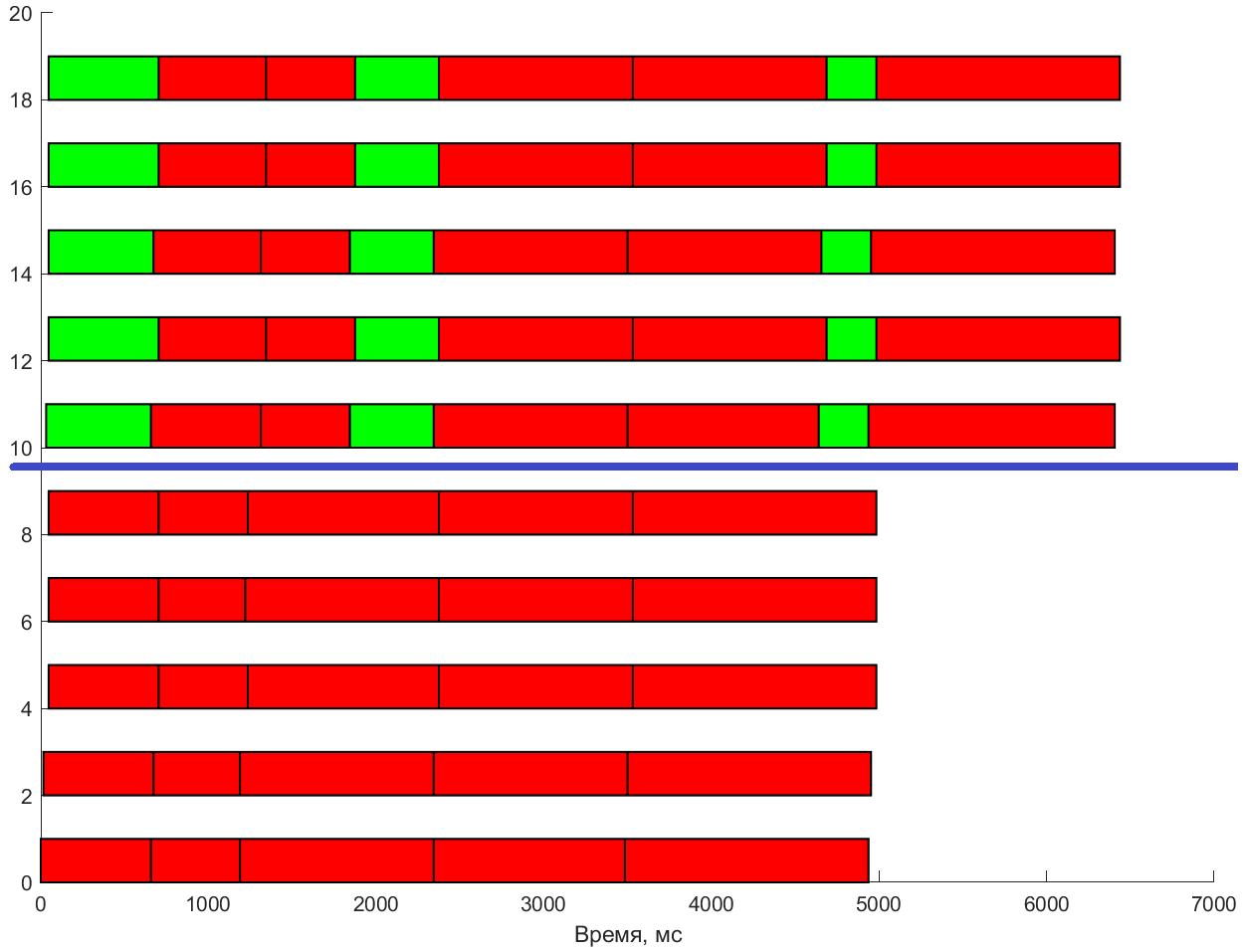

Рисунок 4 – График состояний процессов

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 (если принебречь долями секунды, пока читатель не увидит флажка “страница готова к чтению”) 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.

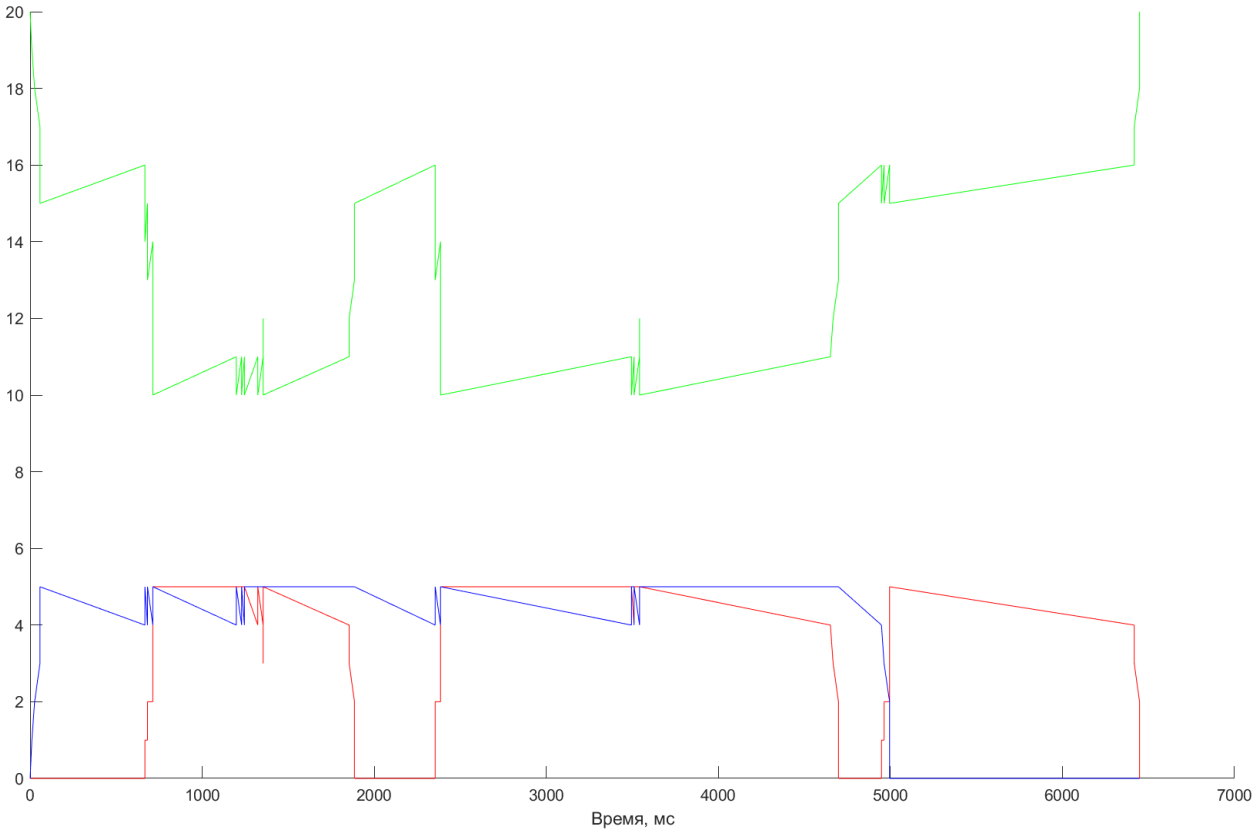

Рисунок 5 – График занятости страниц

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 6 и 7 соответственно.

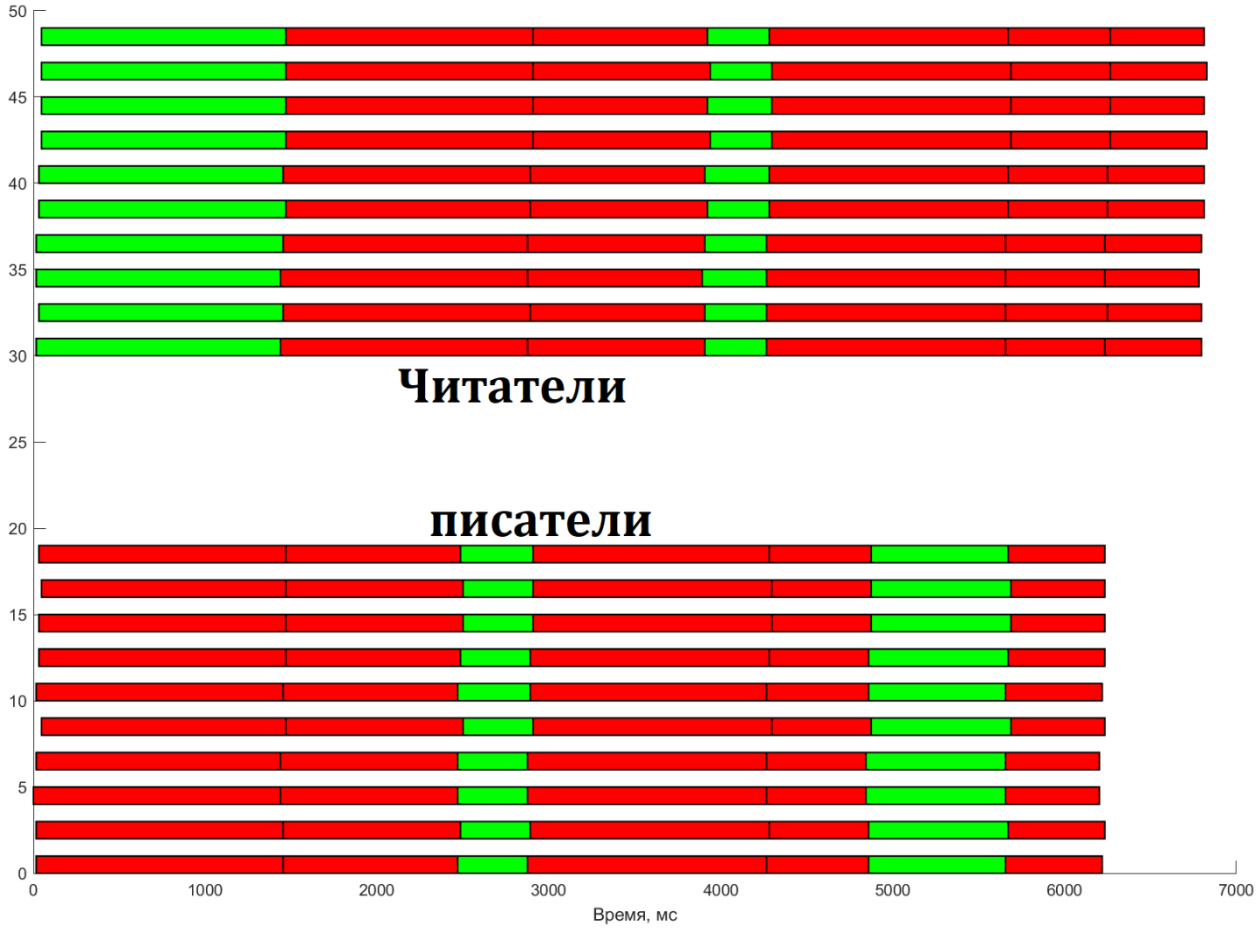

Рисунок 6 – Состояния процессов (10 читателей/писателей)

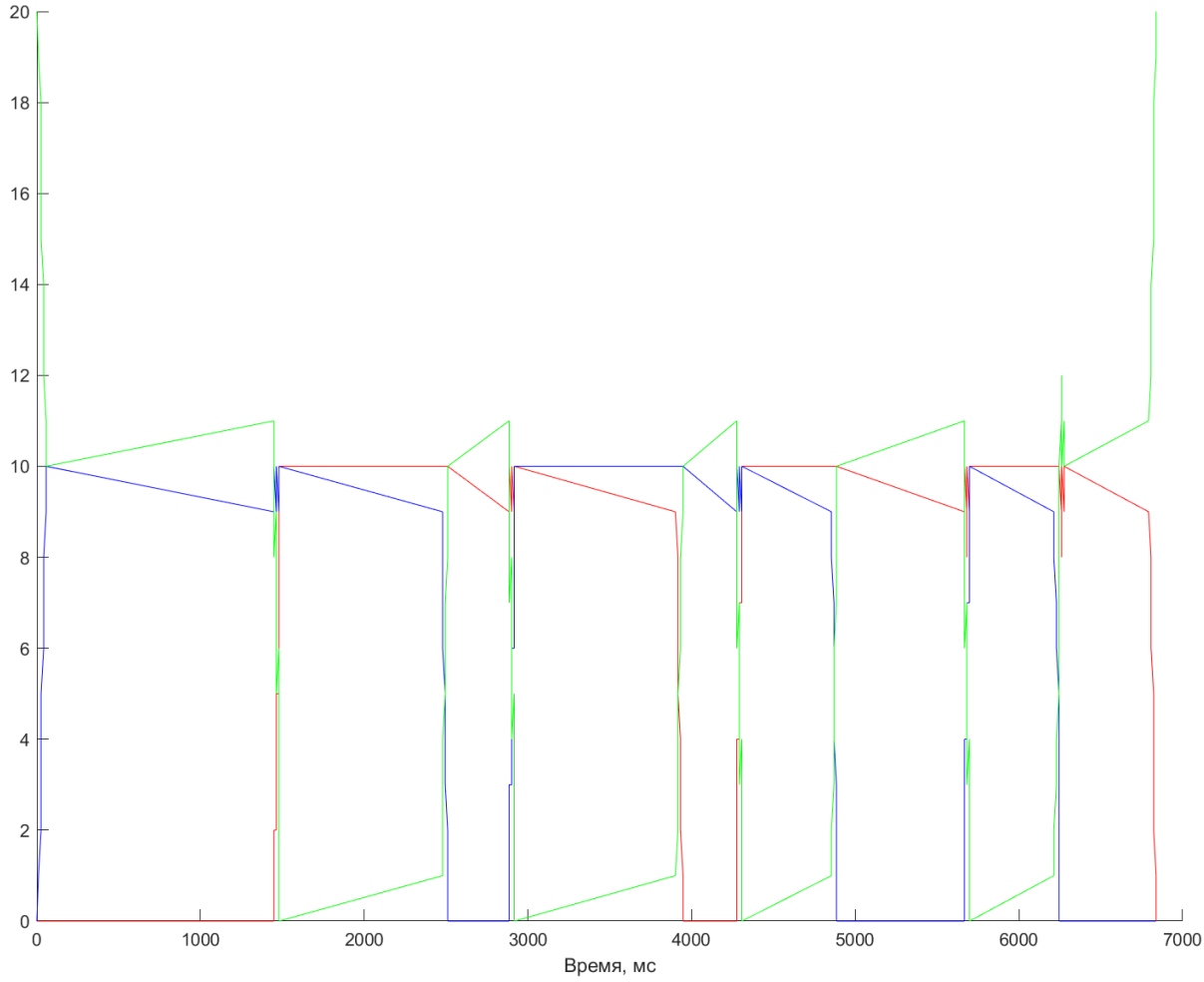

Рисунок 7 – Занятость страниц (10 читателей/писателей)

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:

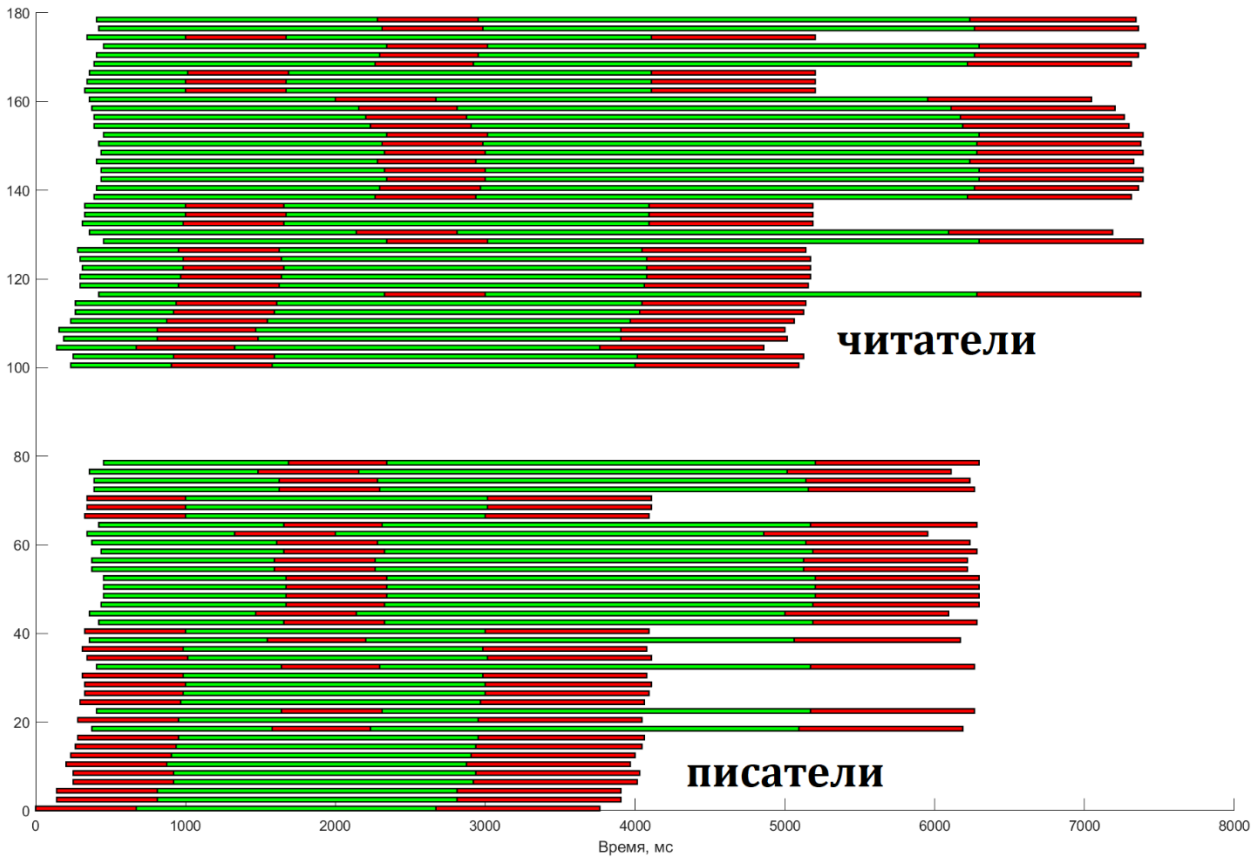

Рисунок 8 – Состояния процессов (40 писателей/читателей)

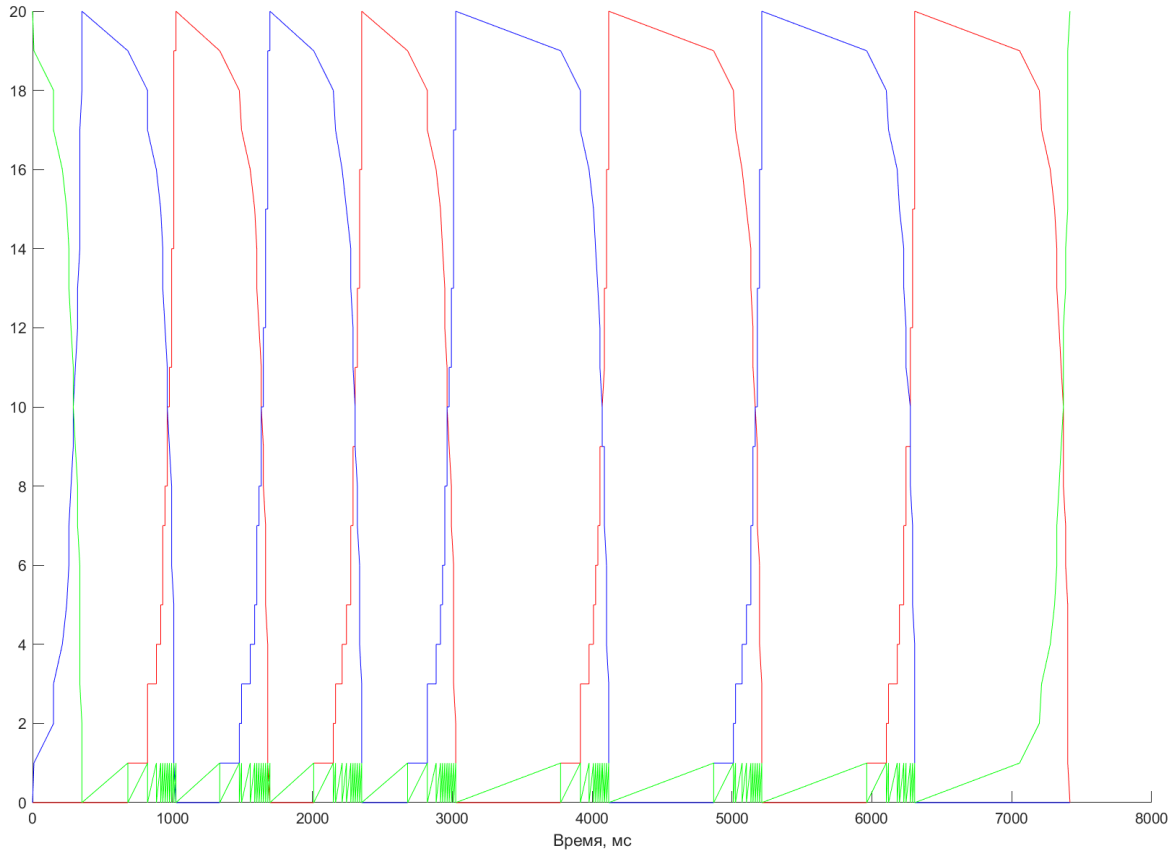

Рисунок 9 – Занятость страниц (40 писателей/читателей)

Вывод по заданию

Было реализовано решение задачи о читателях-писателях, используя общую память между процессами в виде проецируемого файла (некоторое число буферных страниц, в данном случае равного 20), позволяющую обмениваться информацией между писателями и читателями. Для того, чтобы синхронизировать работу данных процессов (если страница занята писателем/читателем, то второй процесс не может забрать ее для чтения/записи), были использованы объекты синхронизации.

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 10.

```
1. Create named pipe
2. Connect client to pipe
3. Disconnect client from pipe
4. Create an event
5. Write to pipe
6. Close pipe
7. Delete event
0. Close the program
>>
```

Рисунок 10 – Меню приложения “сервер”

При выборе первого пункта пользователю предоставляется возможность назвать именнованный канал. Сам процесс создания канала показан на рисунке 11. Данное название понадобится потом в приложении “клиент”.

```
Input the name of a pipe (128 symbols maximum, must starts with \\.\pipe\): \\.\pipe\lab4pipe
```

Рисунок 11 – 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 12 стоит выбрать пункт 1.

```
1. Connect to a named pipe
2. Read from the pipe
0. Close the program
>>
```

Рисунок 12 – Меню приложения “клиент”

При успешном подключении на стороне сервера появится сообщение, показанное на рисунке 13, а у клиента – сообщение на рисунке 14.

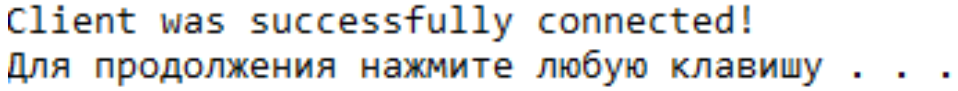

Рисунок 13 – Сообщение об успешном подключении на стороне сервера

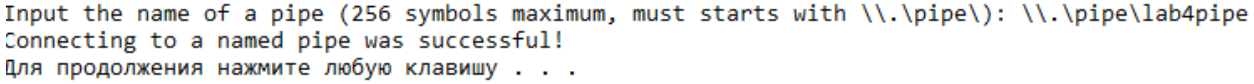

Рисунок 14 – Сообщение об успешном подключении на стороне клиента

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 15.

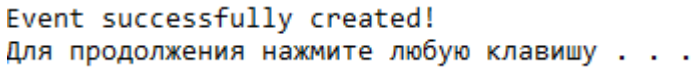

Рисунок 15 – Создание события

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 16 и 17 для сервера и клиента соответственно.

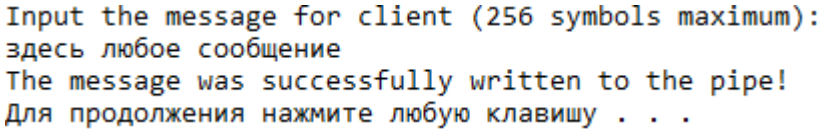

Рисунок 16 – Написание сообщения в канал сервером

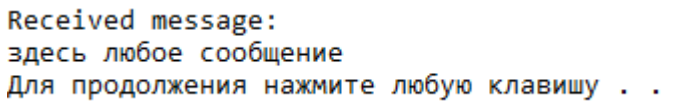

Рисунок 17 – Получение сообщения клиентом

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

Были исследованы инструменты и механизмы взаимодействия процессов в Windows на примере реализации решения задачи о читателях/писателях и с использованием именновах каналов для создания сетевого межпроцессного взаимодействия