МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И.УЛЬЯНОВА (ЛЕНИНА)

Кафедра ВТ

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Операционные системы»
Тема: Межпроцессное взаимодействие

Студент гр. 8307 _____ Репин С.А.

Преподаватель _____ Тимофеев А.В.

Санкт-Петербург
2020

СОДЕРЖАНИЕ

Цель работы 3

Введение 3

1. Реализация решения задачи о читателях-писателях 3

2. Использование именованных каналов для реализации сетевого межпроцессного взаимодействия 11

Вывод 13

ЦЕЛЬ РАБОТЫ

Исследовать инструменты и механизмы взаимодействия процессов в Windows.

ВВЕДЕНИЕ

При выполнении лабораторной работы на языке программирования C стандарта C11 было разработано 2 консольных приложения, соответственно для каждого задания. Исходный код приложения доступен на GitHub [1].

Сборка проектов производится с помощью Powershell-скриптов *build.ps1* (следует создать папку *build* и запускать скрипт из нее). Также потребуется пакет Build Tools for Visual Studio 2019.

1. РЕАЛИЗАЦИЯ РЕШЕНИЯ ЗАДАЧИ О ЧИТАТЕЛЯХ-ПИСАТЕЛЯХ

Программа представляет собой один исполняемый файл, который в зависимости от переданных аргументов командной строки может представляться Читателей, Писателем или Управляющим (который запускает процессы Читателей и Писателей). Имеет следующую структуру:

Файл	Описание
main.c	Точка входа в программу; выбор нужных действия на основании переданных аргументов
config.h	Определение констант
launcher.c	Реализация Управляющего: функций для запуска процессов Читателя и Писателя
error.c	Описание номеров ошибок, а также функции отображения сообщений об ошибках
reader_writer.c	Операции захвата ресурсов (различные файлы, семафоры и мьютексы), Читателя и Писателя

Таблица 1 Описание файлов в проекте для Задания 1

[1] https://github.com/stnrepin/os_labs/tree/master/lab4

Задача заключается в организации совместного доступа набора процессов к одной области памяти для записи и чтения. Те процессы, которые выполняют запись называются Писателями, а те, что выполняют чтение — читателями.

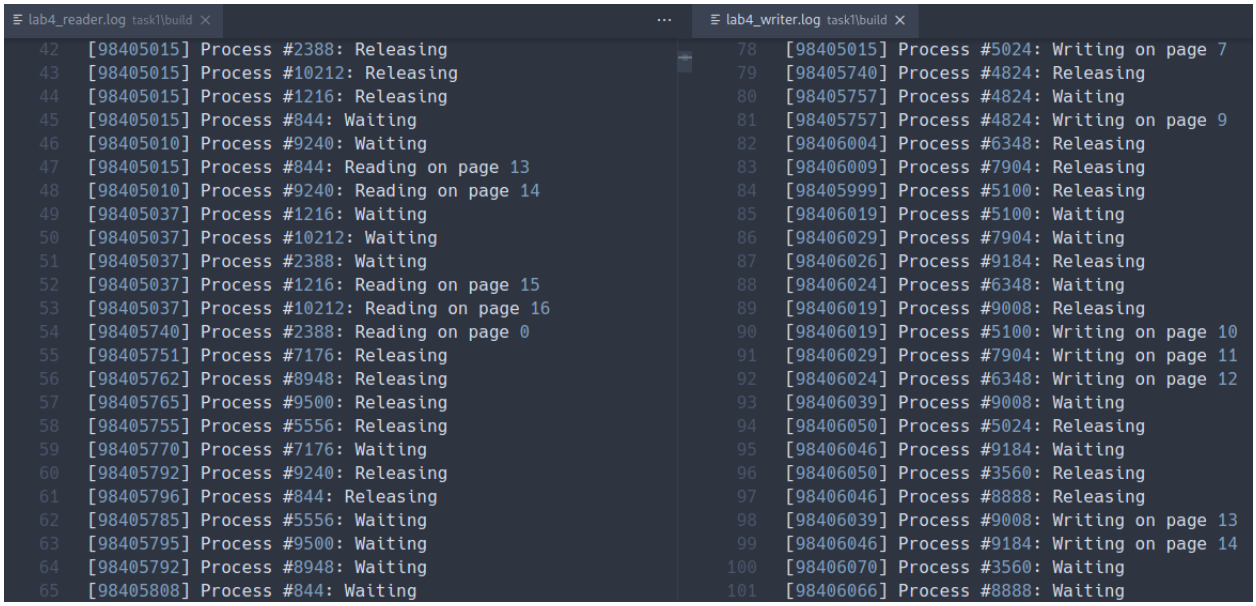
Алгоритм работы читателей следующий:

1. Захватить необходимые ресурсы (открыть объекты семафоров, мьютексов и файлов)
2. Зайти в бесконечный цикл
3. Ожидать разрешения доступа к буферам для читателей (ожидание освобождения семафора читателей, он содержит число занятых буферов)
4. Ожидать разрешения эксклюзивного доступа к буферу (ожидание мьютексов буферов, будет использован первый освободившейся буфер)
5. Выполнить чтение
6. Освободить доступ к буферу
7. Предоставить новое вакантное место для писателя (освободить семафор писателей)

Алгоритм работы писателя следующий:

1. Захватить необходимые ресурсы (открыть объекты семафоров, мьютексов и файлов)
2. Зайти в бесконечный цикл
3. Ожидать разрешения доступа к буферам для писателей (ожидание освобождения семафора писателей, он содержит число свободных буферов)
4. Ожидать разрешения эксклюзивного доступа к буферу (ожидание мьютексов буферов, будет использован первый освободившейся буфер)
5. Выполнить запись
6. Освободить доступ к буферу
7. Предоставить новое вакантное место для читателя (освободить семафор читателей)

В итоге выполнения программы при 9 Читателях, 9 Писателях и 17 буферах были получены следующие логи:



lab4_reader.log task1\build

```
[98405015] Process #2388: Releasing
[98405015] Process #10212: Releasing
[98405015] Process #1216: Releasing
[98405015] Process #844: Waiting
[98405010] Process #9240: Waiting
[98405015] Process #844: Reading on page 13
[98405010] Process #9240: Reading on page 14
[98405037] Process #1216: Waiting
[98405037] Process #10212: Waiting
[98405037] Process #2388: Waiting
[98405037] Process #1216: Reading on page 15
[98405037] Process #10212: Reading on page 16
[98405740] Process #2388: Reading on page 0
[98405751] Process #7176: Releasing
[98405762] Process #8948: Releasing
[98405765] Process #9500: Releasing
[98405755] Process #5556: Releasing
[98405770] Process #7176: Waiting
[98405792] Process #9240: Releasing
[98405796] Process #844: Releasing
[98405785] Process #5556: Waiting
[98405795] Process #9500: Waiting
[98405792] Process #8948: Waiting
[98405808] Process #844: Waiting
```

lab4_writer.log task1\build

```
[98405015] Process #5024: Writing on page 7
[98405740] Process #4824: Releasing
[98405757] Process #4824: Waiting
[98405757] Process #4824: Writing on page 9
[98406004] Process #6348: Releasing
[98406009] Process #7904: Releasing
[98405999] Process #5100: Releasing
[98406019] Process #5100: Waiting
[98406029] Process #7904: Waiting
[98406026] Process #9184: Releasing
[98406024] Process #6348: Waiting
[98406019] Process #9008: Releasing
[98406019] Process #5100: Writing on page 10
[98406029] Process #7904: Writing on page 11
[98406024] Process #6348: Writing on page 12
[98406039] Process #9008: Waiting
[98406050] Process #5024: Releasing
[98406046] Process #9184: Waiting
[98406050] Process #3560: Releasing
[98406046] Process #8888: Releasing
[98406039] Process #9008: Writing on page 13
[98406046] Process #9184: Writing on page 14
[98406070] Process #3560: Waiting
[98406066] Process #8888: Waiting
```

Рис. 1.1 Фрагмент файлов журналов Читателей (слева) и Писателей (справа)

По журнальным файлам видно, что в каждый момент времени одновременно работает 17 процессов, что соответствует числу буферов. А доступ к буферу захватывается Писателем или Читателем сразу же, как он освобождается.

Приведем графики смены состояний 5 процессов Читателей и 5 процессов Писателей (при 17 буферах). На каждом графике изображена последовательность состояний

$$\{init, wait, read, release, wait, read, wait, read, release\}$$

По вертикальной оси расположены названия состояний процесса, а по горизонтальной – время (здесь и далее 4 старшие цифры удалены, так как везде одинаковые); точками обозначены моменты перехода в состояние. Заметим, что все графики сильно друг на друга похожи, отличаясь только абсолютными значениями.

Графики Читателей:

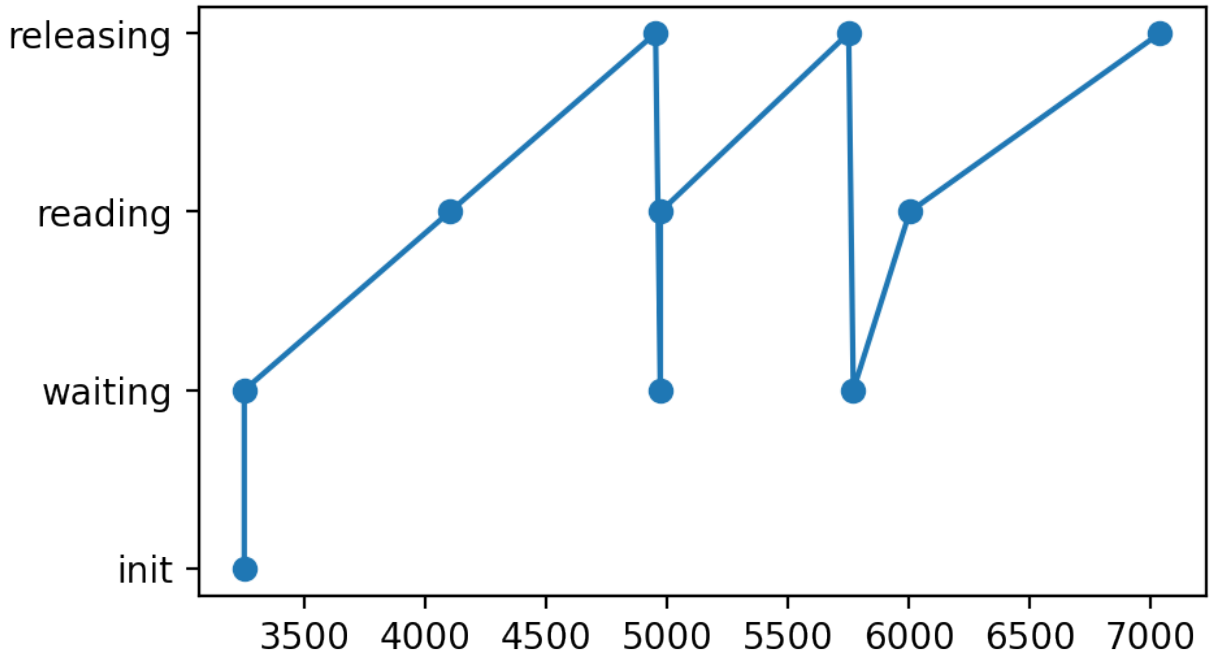


Рис. 1.2

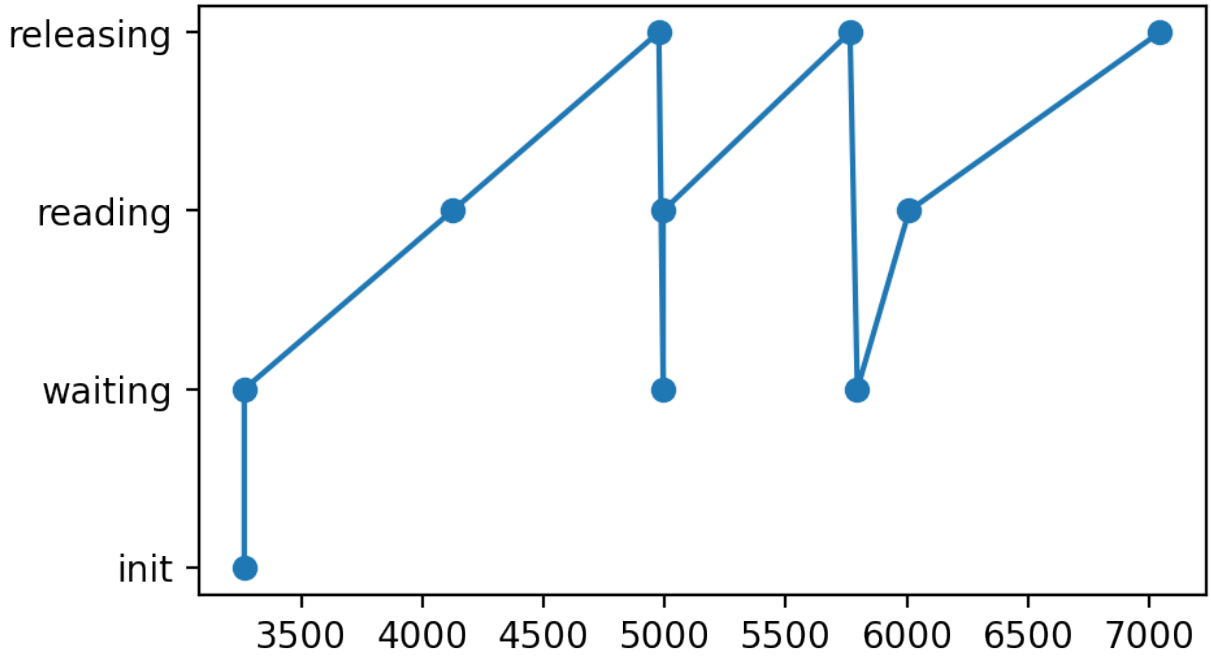


Рис. 1.3

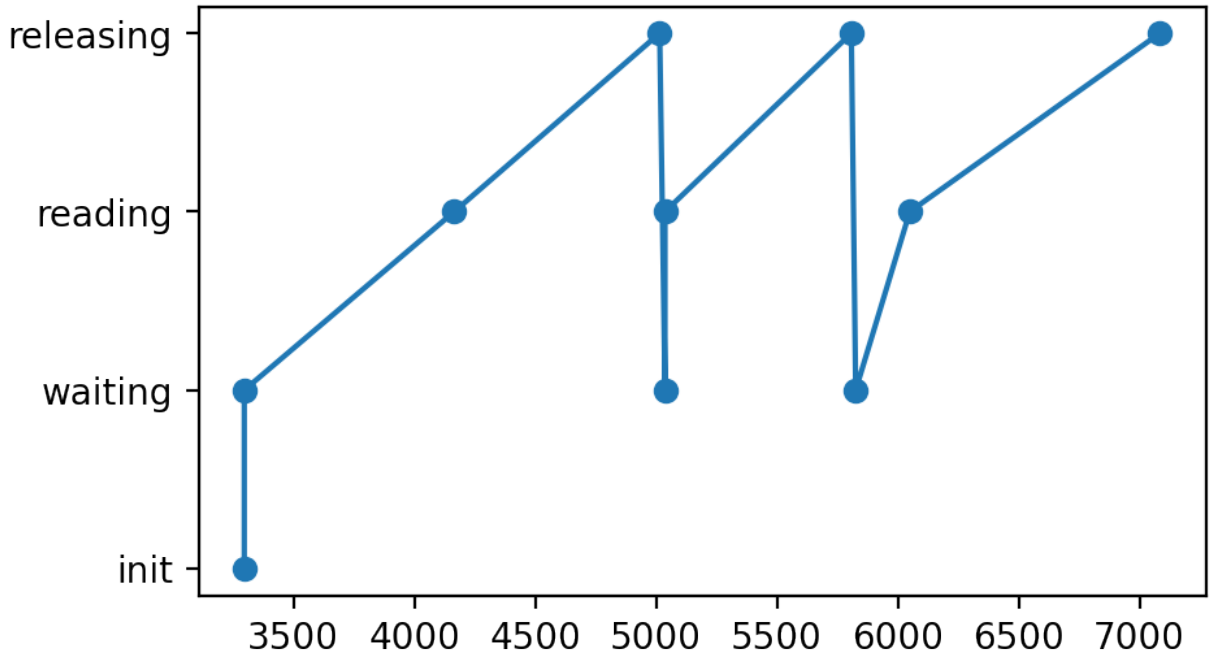

releasing
reading
waiting
init
3500
4000
4500
5000
5500
6000
6500
7000

Рис. 1.4

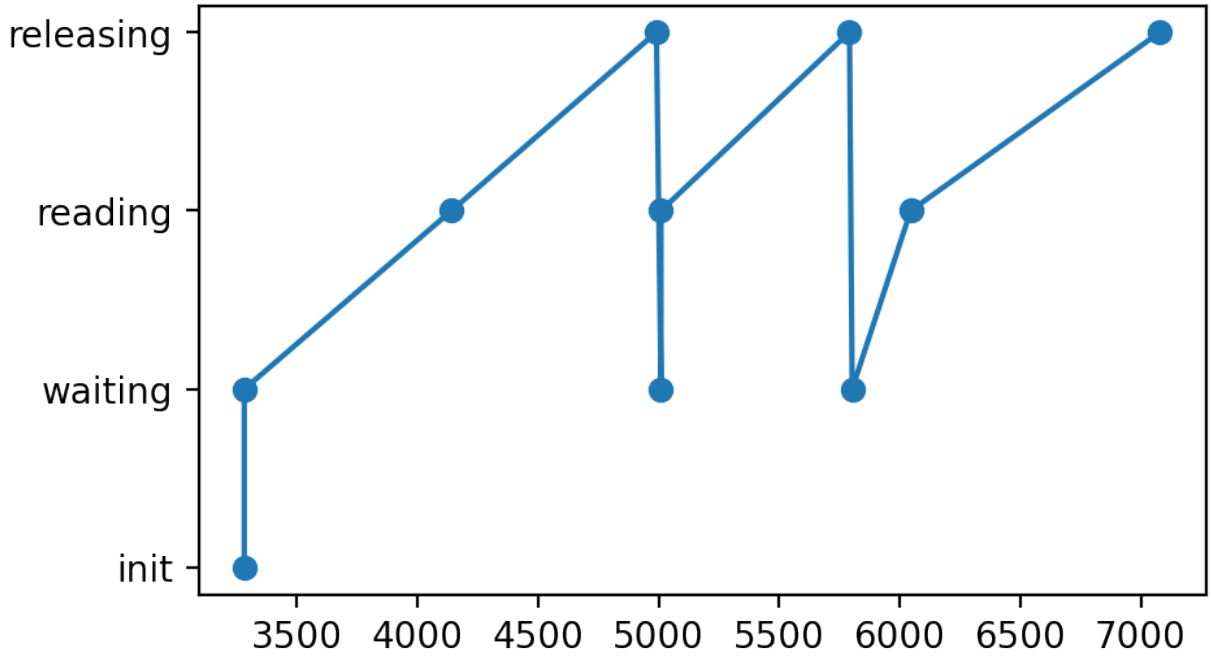

releasing
reading
waiting
init
3500
4000
4500
5000
5500
6000
6500
7000

Рис. 1.5

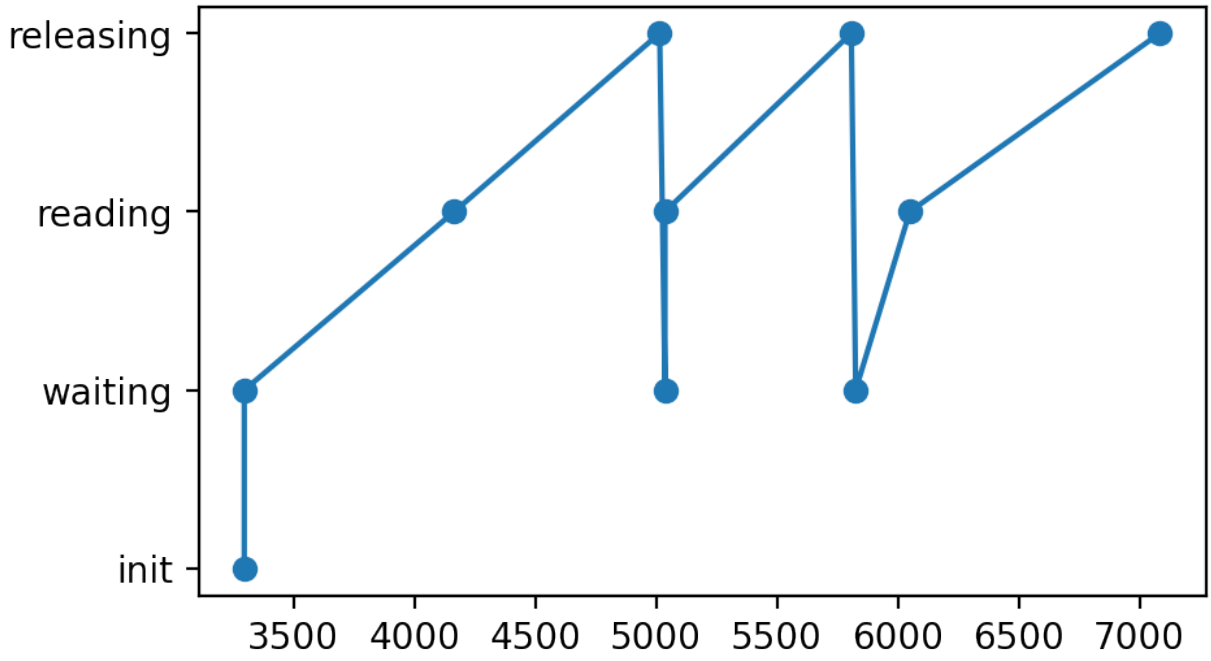


Рис. 1.6

Графики Писателей:

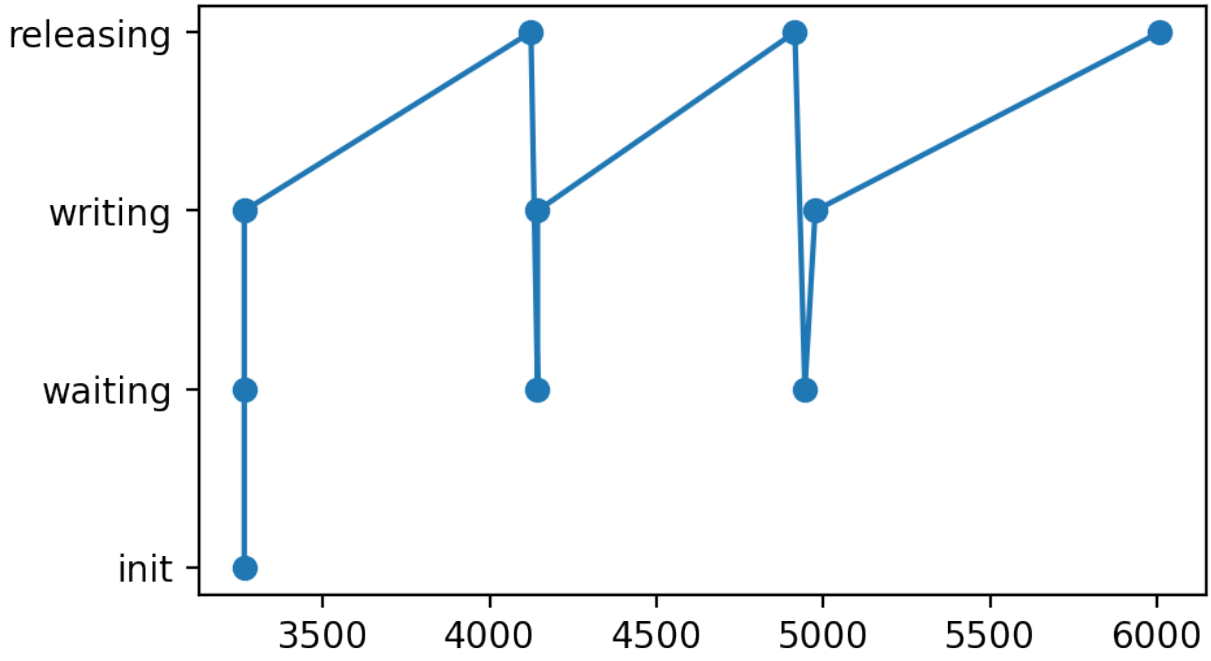


Рис. 1.7

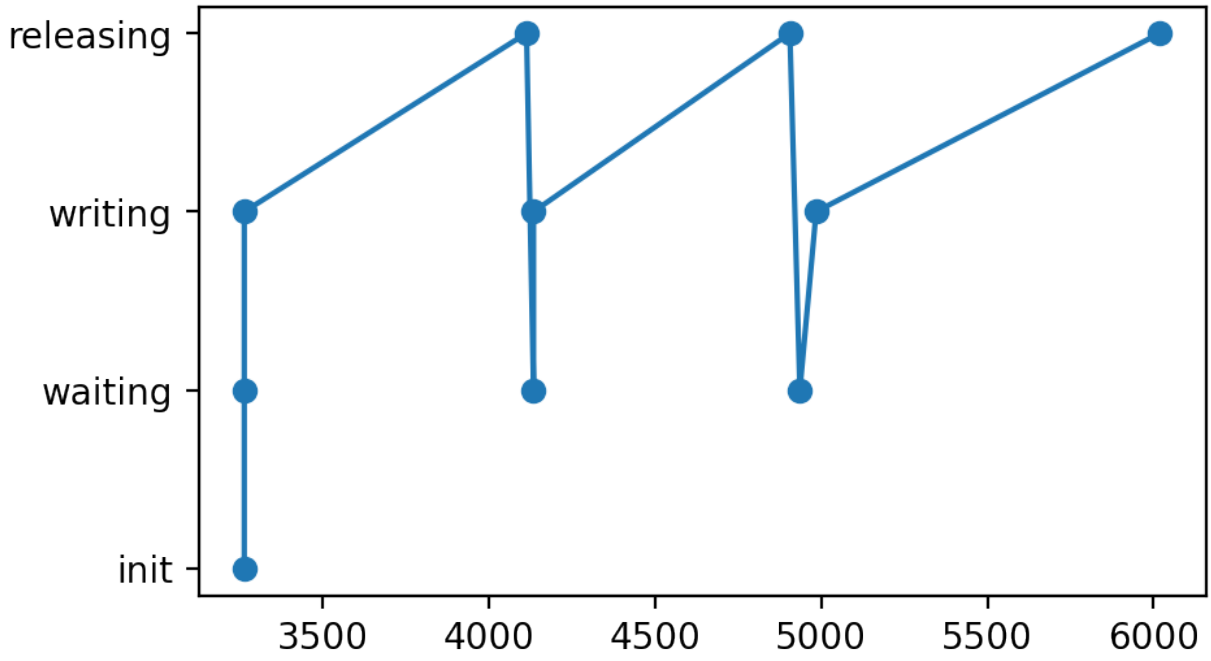

releasing
writing
waiting
init
3500
4000
4500
5000
5500
6000

Рис. 1.8

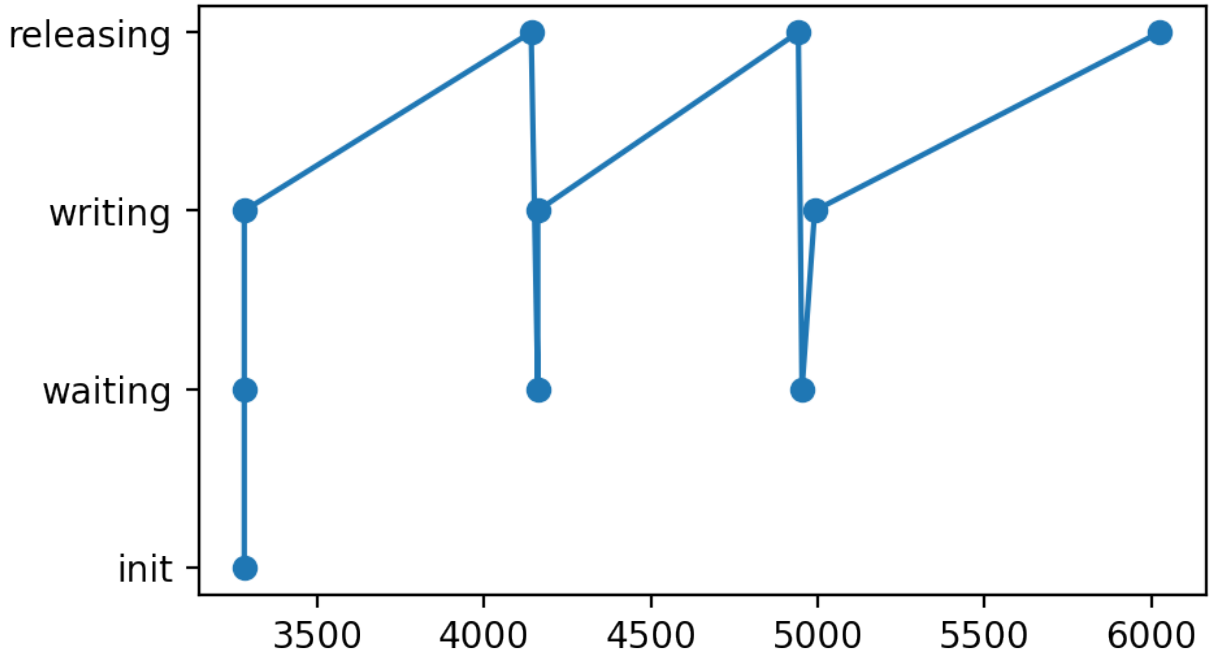

releasing
writing
waiting
init
3500
4000
4500
5000
5500
6000

Рис. 1.9

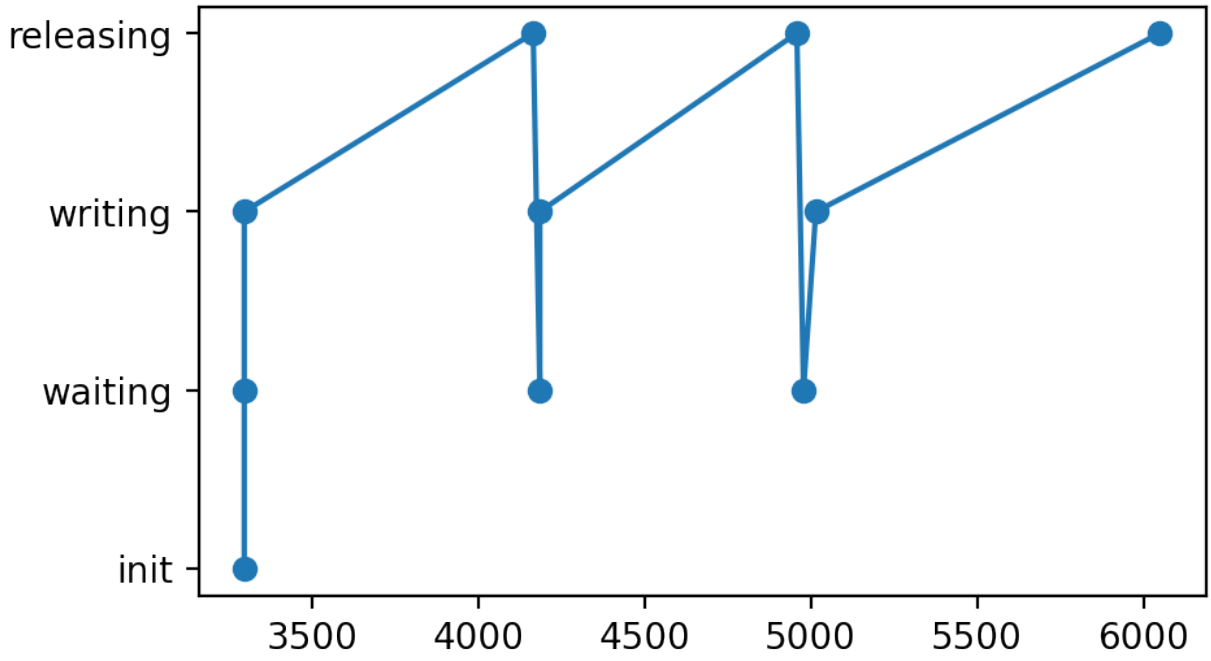


Рис. 1.10

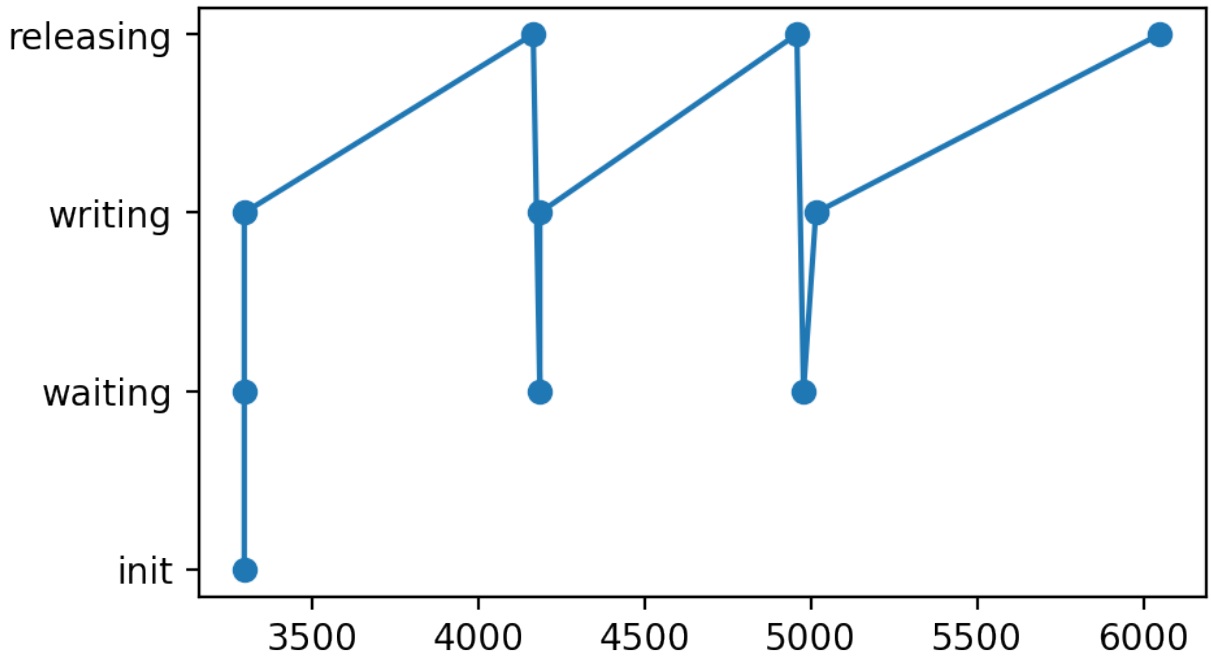


Рис. 1.11

Изобразим графики занятости некоторых произвольных буферов (1, 5, 7) при тех же установках, что и ранее.

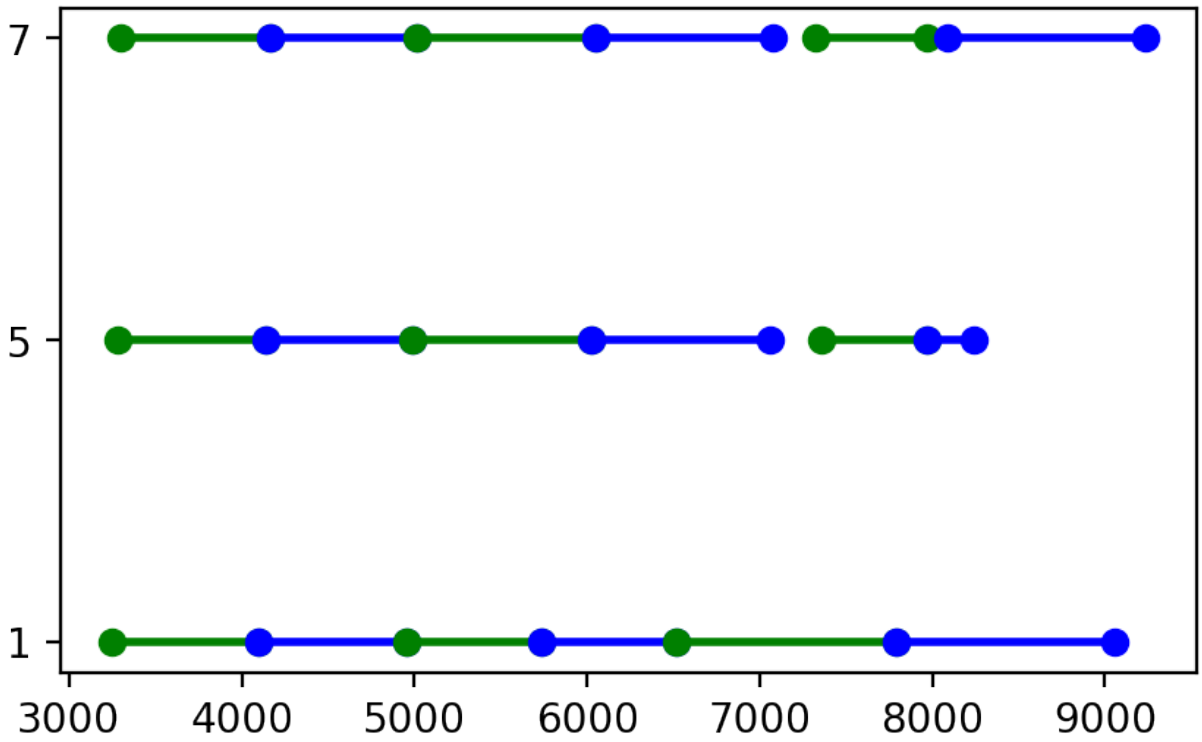


Рис. 1.12

2. ИСПОЛЬЗОВАНИЕ ИМЕНОВАННЫХ КАНАЛОВ ДЛЯ РЕАЛИЗАЦИИ СЕТЕВОГО МЕЖПРОЦЕССНОГО ВЗАИМОДЕЙСТВИЯ

В данном задании создается консольное приложение, работающее в двух режимах:

- Сервер. Создает именованный канал, ожидает подключения клиента, производит запись данных в канал
- Клиент. Подключается к именованному каналу, асинхронно считывает данные и отображает их на экран

Пример выполнения программы приведен на рисунке ниже.

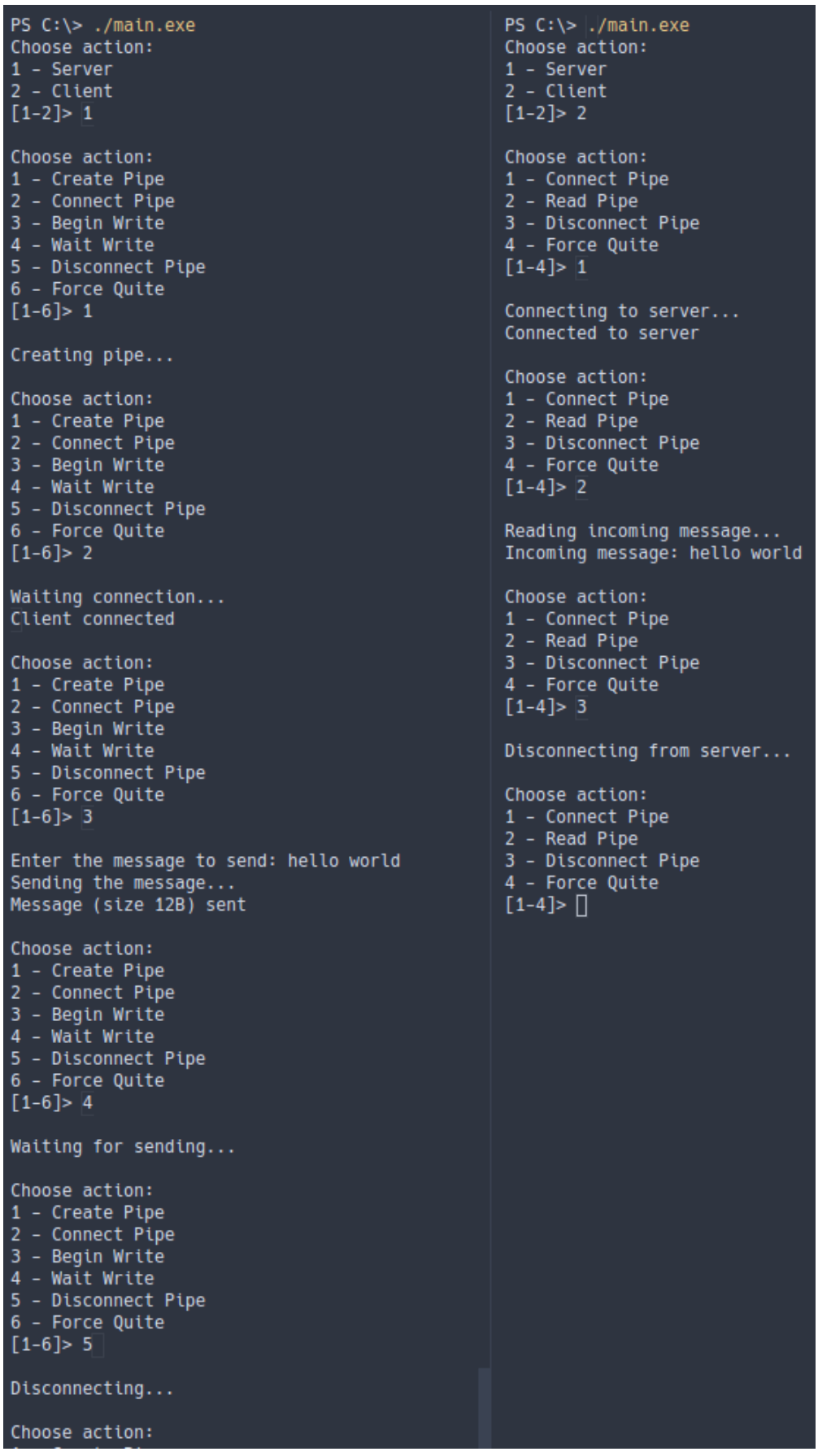


Рис. 2.1 Передача сообщения от сервера (слева) клиенту (справа)

ВЫВОД

В ходе выполнения данной лабораторной работы были получены знания и практические навыки организации взаимодействия процессов с помощью WinAPI. Для этой цели были использованы как базовые элементы — примитивы синхронизации — семафоры и мьютексы, так и более высокоуровневые механизмы — именованные каналы.

В первом задании было написано решение задачи о читателях-писателях, здесь применялись семафоры и мьютексы для обеспечения эксклюзивного доступа к общей для процессов памяти. Принцип их работы крайне схож: они оба ограничивают количество потоков, имеющих общий доступ к критической секции, только у семафора это число является параметром, задающимся при его создании, а у мьютекса оно всегда равно 2 (поэтому мьютекс еще называют бинарным семафором). В задаче с помощью мьютексов делается так, что только один процесс имеет доступ к буферу в конкретный момент времени (буфер может быть разблокирован, а может быть заблокирован), а с помощью семафоров — так, что чтение данных и их запись между собой не конфликтует (сначала происходит запись, а потом чтение). Это все важно для корректности работы программы, так как в противном случае возникает состояние гонки, считающееся неопределенным поведением, или чтение до записи, что само по себе неправильно.

Рассматривая переходы процессов из одного состояния в другое видно, что для всех них наблюдается схожая картина: каждый раз процесс ждет освобождения буфера, производит над ним чтение или запись и снова возвращается в очередь, передавая буфер читателю или писателю. Первые операции чтения/записи происходят с минимальным временем ожидания, которое затем увеличивается из-за роста количества работающих Читателей и Писателей. Причем заметим, что по графикам занятости буферов видно, что над буфером постоянно совершаются чередующиеся операции чтения или записи, которые никогда не пересекаются, а сам буфер утилизируется практически непрерывно.

Во втором задании применяются именованные каналы для налаживания общения между различными процессами. Главным преимуществом по сравнению с использованием примитивов является то, что код программы становится намного проще как с точки зрения написания, так и с точки зрения чтения. Как известно, ошибки связанные с многопоточным и многопроцессным программированием тяжело поддаются диагностике, поэтому упрощение такого кода очень важно.