Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

”Национальный исследовательский университет ИТМО“

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА И РЕАЛИЗАЦИЯ МЕТОДОВ ЭФФЕКТИВНОГО ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ

Автор Губарев Владимир Юрьевич _____

Направление подготовки 09.04.04 ”Программная инженерия“

Квалификация Магистр

Руководитель Косяков М.С., к.т.н. _____

Санкт-Петербург, 2020 г.

Обучающийся Губарев В.Ю.
Группа Р42111 Факультет/институт/кластер ПИиКТ

Направленность (профиль), специализация
Информационно-вычислительные системы, Интеллектуальные системы

ВКР принята «_____» _____ 20___ г.

Оригинальность ВКР _____%

ВКР выполнена с оценкой _____

Дата защиты «_____» _____ 20___ г.

Секретарь ГЭК Болдырева Е.А. _____

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

”НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО“

УТВЕРЖДАЮ

Руководитель ОП

доцент, д.т.н. Бессмертный И.А. _____

«_____» _____ 20___ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Обучающийся Губарев В.Ю.

Группа P42111 **Факультет/институт/кластер** ПИиКТ

Квалификация Магистр

Направление подготовки 09.04.04 ”Программная инженерия“

Направленность (профиль) образовательной программы

Информационно-вычислительные системы

Специализация Интеллектуальные системы

Тема ВКР Разработка и реализация методов эффективного взаимодействия процессов в распределенных системах

Руководитель Косяков М.С., к.т.н., доцент ФПИиКТ

2 Срок сдачи студентом законченной работы «_____» _____ 20___ г.

3 Техническое задание и исходные данные к работе

Требуется разработать и реализовать эффективные методы межпроцессного взаимодействия в пределах одного физического узла. Межпроцессное взаимодействие как с локальными, так и с удаленными процессами должно осуществляться через единый программный интерфейс. Интерфейс должен автоматически выбирать наиболее эффективный метод межпроцессного взаимодействия и скрывать реализацию от пользователя.

4 Содержание выпускной работы (перечень подлежащих разработке вопросов)

а) Обзор предметной области и постановка цели работы.

б) Разработка и реализация методов эффективного взаимодействия процессов.

в) Экспериментальное исследование и обработка результатов.

5 Перечень графического материала (с указанием обязательного материала)

а) Гистограммы временной задержки на передачу данных для разработанных методов межпроцессного взаимодействия.

б) Принципиальные схемы разработанных методов межпроцессного взаимодействия.

6 Исходные материалы и пособия

а) Косяков М.С. Введение в распределенные вычисления. Учебное пособие / М.С. Косяков. – СПб: СПбГУ ИТМО, 2014. – 155 с.

б) Schmidt D.C. et al. Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects. – John Wiley & Sons, 2013. – Т. 2.

7 Дата выдачи задания «_____» _____ 20___ г.

Руководитель ВКР _____

Задание принял к исполнению _____«_____» _____ 20___ г.

Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

”НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО“

АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Обучающийся Губарев Владимир Юрьевич

Наименование темы ВКР: Разработка и реализация методов эффективного взаимодействия процессов в распределенных системах

Наименование организации, в которой выполнена ВКР Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования: уменьшение временной задержки на передачу данных между процессами в пределах одного физического узла путем разработки и применения методов эффективного межпроцессного взаимодействия.

2 Задачи, решаемые в ВКР:

а) рассмотреть существующие методы межпроцессного взаимодействия, доступные при взаимодействии процессов, находящихся на одном физическом узле;

б) произвести анализ и отбор методов межпроцессного взаимодействия для реализации новых методов межпроцессного взаимодействия;

в) разработать и реализовать эффективные методы межпроцессного взаимодействия;

г) экспериментально исследовать полученные методы межпроцессного взаимодействия.

3 Число источников, использованных при составлении обзора:

4 Полное число источников, использованных в работе: 17

5 В том числе источников по годам:

Отечественных			**Иностранных**		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
2	0	0	2	5	8

6 Использование информационных ресурсов Internet: да, число ресурсов: 5

7 Использование современных пакетов компьютерных программ и технологий:

Пакеты компьютерных программ и технологий	**Раздел работы**
LaTeX	Весь текст диссертации и сопроводительные документы
C++17 (“International Standard ISO/IEC 14882:2014(E) Programming Language C++”)	Раздел 2.4.3.1, Приложение А
LTTng	Глава 3

8 Краткая характеристика полученных результатов

Разработано семейство новых методов межпроцессного взаимодействия в пределах одного физического узла, показавших меньшую временную задержку на передачу данных, чем разработанные ранее.

9 Гранты, полученные при выполнении работы

Отсутствуют.

10 Наличие публикаций и выступлений на конференциях по теме работы

1 *Губарев В. Ю.* Реализация методов эффективного взаимодействия процессов в распределенных системах // Сборник тезисов докладов конгресса молодых ученых. Электронное издание. — Университет ИТМО, 2020.

Обучающийся Губарев В.Ю. _____

Руководитель Косяков М.С. _____

«____» _____ 20__ г.

ABSTRACT

Development and implementation of efficient inter-process communication methods for distributed systems

Nowadays distributed systems are widely spreaded. They are usually designed to work in various environments as a set of cooperating processes. At the same time capabilities of modern hardware allow to deploy groups of that processes within a single machine in order to achieve better performance. In this case efficient inter-process communication (IPC) methods become a crucial element of high-performance distributed systems.

The present work is focused on developing efficient IPC methods. Based on the most efficient IPC in Linux, shared memory and futex, it introduces new methods of low-latency IPC. They are transparently provided via a generic interface. The interface automatically and transparently for programmer uses TCP to communicate over network with remote processes and low-latency shared memory-based method for local processes.

Proposed methods show significantly lower latency with local processes than TCP-based without any additional difficulties for programmer.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Объектом исследования являются методы межпроцессного взаимодействия.

Предметом исследования является временная задержка на передачу данных между процессами распределенной системы в пределах одного физического узла.

Цель работы – уменьшение временной задержки на передачу данных между процессами в пределах одного физического узла путем разработки и применения методов эффективного межпроцессного взаимодействия.

В настоящей работе поставлены следующие **задачи**:

- рассмотреть существующие методы межпроцессного взаимодействия, доступные при взаимодействии процессов, находящихся на одном физическом узле;
- произвести анализ и отбор методов межпроцессного взаимодействия для реализации новых методов межпроцессного взаимодействия;
- разработать и реализовать эффективные методы межпроцессного взаимодействия;
- экспериментально исследовать полученные методы межпроцессного взаимодействия.

Актуальность исследования.

Для некоторых систем эффективное межпроцессное взаимодействие является критически важной частью их работы. Требование по минимизации времени обслуживания заявок может напрямую следовать из области применения системы, как в случае с системами для алгоритмической торговли на финансовых рынках. Обслуживание заявок множеством логически связанных процессов может быть существенно ускорено при размещении таких процессов на одном физическом узле. Современные процессоры с количеством с десятками вычислительных ядер могут обеспечить такую конфигурацию нужными ресурсами. Это позволяет использовать более эффективные методы межпроцессного взаимодействия, а именно методы на основе разделяемой памяти [4]. Эффективные методы межпроцессного взаимодействия могут использоваться для связи виртуальных машин или контейнеров в пределах машины-хозяина [10, 16]. Для связи программных модулей, исполняющихся в разных процессах для обеспечения отказоустойчи-

вости за счет изоляции процессов на уровне ОС. Для высокопроизводительных вычислений, таких как анализ научных данных или прогнозирование погоды.

При разработке сложной многокомпонентной распределенной системы программисту необходимо сосредоточиться на логике и корректности работы самой системы. В то время как методы межпроцессного взаимодействия должны быть для него прозрачны. Этого можно достичь, используя единый унифицированный интерфейс для межпроцессного взаимодействия. Это упрощает разработку, снимает необходимость сложного управления ресурсами для межпроцессного взаимодействия. А также позволяет автоматически использовать наиболее подходящие методы межпроцессного взаимодействия для данных пространственных конфигураций процессов, что может повысить эффективность выполнения некоторых задач этой системой.

Таким образом, разработка и реализация эффективных методов межпроцессного взаимодействия и интерфейса для автоматического доступа к наиболее подходящим из них необходима и обоснована. Посредством этого интерфейса программист прозрачно для себя использует методы межпроцессного взаимодействия на основе разделяемой памяти при взаимодействии с локальными процессами без необходимости перекомпиляции программы. Но поскольку зачастую нельзя разместить всю систему на одном, даже очень производительном, сервере используется TCP при взаимодействии с процессами на других физических узлах.

Методы исследования включают в себя анализ существующих методов межпроцессного взаимодействия, экспериментальное исследование разработанных методов межпроцессного взаимодействия и методы математической статистики для обработки экспериментальных данных.

Средства исследования:

- язык программирования C++, компилятор *Clang 6.0.1*, стандартная библиотека C++ *libstdc++*;
- Библиотека Boost.Interprocess [5] для управления разделяемой памятью;
- система трассировки событий [2] на основе инструмента LTTng [9].

Научная новизна заключается в предложенных новых методах эффективного межпроцессного взаимодействия в пределах одного физического узла, которые не описаны в существующих исследованиях.

Положения, выносимые на защиту

Методы межпроцессного взаимодействия:

— через очередь в разделяемой памяти с оповещением о появлении данных в очереди через мультиплексор в разделяемой памяти и обслуживанием соединений по модели ”Лидер/Последователи“ с ожиданием сигналов потоком в режиме сна на futex;
— через очередь в разделяемой памяти с оповещением о появлении данных в очереди через мультиплексор в разделяемой памяти и обслуживанием соединений по модели ”Полусинхронный/Полуреактивный“ с ожиданием сигналов потоком в режиме сна на futex;
— через очередь в разделяемой памяти с оповещением о появлении данных в очереди через мультиплексор в разделяемой памяти и обслуживанием соединений по модели ”Лидер/Последователи“ с ожиданием сигналов потоком в режиме активного опроса мультиплексора.

Апробация результатов.

Основные результаты работы были представлены на IX Конгрессе Молодых Ученых.

Предложенные методы межпроцессного взаимодействия на основе разделяемой памяти подготовлены к использованию в платформе для торговли на финансовых рынках Tbricks от компании Itiviti.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЦЕЛИ РАБОТЫ

ГЛАВА 2. РАЗРАБОТКА И РЕАЛИЗАЦИЯ МЕТОДОВ ЭФФЕКТИВНОГО ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ

2.1. Интерфейс доступа к разрабатываемым методам межпроцессного взаимодействия

Существенная часть межпроцессного взаимодействия - это то, как программист его использует и как сделать это удобным процессом. Интерфейс сокетов в ОС Linux хорошо для этого, так как использует механизм файловых дескрипторов , а значит:

— имеет стандартные примитивы чтения и записи: системные вызовы *read/readv* и *write/writev*;
— позволяет мультиплексировать множество соединений для одновременного отслеживания посредством системных мультиплексоров: *select/poll/epoll*.

Удобный и унифицированный интерфейс может объединять под собой совершенно разные методы межпроцессного взаимодействия прозрачным для программиста образом. Программист зачастую не заинтересован в тонкостях передачи данных, ему необходимо реализовывать и совершенствовать пользовательскую логику приложения, однако, поскольку процессы распределенной системы работают вместе над одними задачами, то ему важна и эффективность межпроцессного взаимодействия.

Необходима возможность:

— предоставлять другим процессам возможность для подключения и подключаться к другим процессам;
— принимать и передавать данные.

Для первого пункта хорошо подходит семейство шаблонов сетевого программирования Acceptor и Connector [12], которые при необходимости создают экземпляры пользовательских обработчиков соединений, реализующих интерфейс, представленный на Листинге 1.

Данный интерфейс соответствует асинхронному событийному шаблону проектирования приложений. В этом шаблоне обработчик соединения является пассивной сущностью, в которую события доставляются посредством вызова метода *handle_message(const Message &)*. Во-первых, использование этого шаблона позволяет разделить пользовательскую логику и нижележащие методы межпро-

Листинг 1 – Интерфейс пользовательского обработчика соединений на C++

```
class ISession
{
public:
    virtual int handle_message(const Message & message) = 0;
    virtual int send(const Message & message) = 0;
    virtual int handle_close() = 0;
};
```

цессного взаимодействия, работающие по произвольным алгоритмам. Следовательно, это упрощает пользовательскую логику. Во-вторых, позволяет обрабатывать множество соединений числом потоков, не равным количеству этих соединений, так как обработчики соединения нуждаются в потоке для своего выполнения только когда необходимо обработать событие. Для отправки сообщений программисту предлагается использовать примитив *send(const Message &)*.

Данный интерфейс позволяет скрыть от программиста тонкости работы межпроцессного взаимодействия. Например, отслеживание готовности множества TCP-соединений к передаче или приему данных, состояние соединений. Или то, что взаимодействие происходит через разделяемую память.

2.2. Базовый метод межпроцессного взаимодействия на основе TCP

TCP – это наиболее широко-используемый и универсальный метод межпроцессного взаимодействия. Протокол гарантирует, что сообщения будут получены в том же количестве и порядке, в котором они были отправлены. Посредством интерфейса сокетов он позволяет взаимодействовать как с локальными, так и с удаленными процессами распределенной системы. Более того, сторонние системы тоже зачастую предоставляют возможность именно подключения по TCP.

Кроме распространенности и удобства для программиста, предоставляют очень важную возможность отслеживания времени жизни соединения. Это важно, так как случае краха процесса операционная система в ходе освобождения системных ресурсов также закроет все TCP-соединения процесса и другие стороны межпроцессного взаимодействия смогут корректно обработать закрытие соединений.

2.2.1. Мультиплексирование соединений

Межпроцессное взаимодействие посредством TCP позволяет использовать системные мультиплексоры событий *select/poll/epoll* для отслеживания событий в множестве TCP-соединений: наличия данных, готовности канала к передаче, закрытия соединения. В системах с большим количеством неактивных соединений лучше всего себя показывает системный мультиплексор *epoll*, при наличии небольшого количества активных соединений большой разницы между ними не наблюдается [3].

Классическим подходом при разработке сетевых приложений является применение шаблона Реактор [14, 17] для диспетчеризации множества соединений. В настоящей работе используется реализаций шаблона Реактор из библиотеке ACE [15], работающая с системным мультиплексором событий *select*.

Реактор – это объект, в котором регистрируются обработчики событий и которые Реактор вызывает посредством вызовов их интерфейсных методов. Сокращенный интерфейс обработчика событий из библиотеки ACE приведен в листинге 2.

Листинг 2 – Интерфейс низкоуровневого обработчика соединений на C++

```
class ACE_Event_Handler
{
public:
    virtual int handle_input(ACE_HANDLE fd) = 0;
    virtual int handle_output(ACE_HANDLE fd) = 0;
    virtual int handle_close(ACE_Handle fd, ACE_Reactor_Mask
   close_mask) = 0;
};
```

2.2.2. Обслуживание активных соединений

2.2.3. Динамическое конфигурирование соединений

2.3. Применение разделяемой памяти для передачи данных

Однако, недостаточно разместить очередь в разделяемой памяти и отправлять в нее сообщения. Кроме контроля времени жизни соединения, существенный вопрос в данном методе: кто, как и когда будет принимать сообщения. На одном физическом узле может находится множество взаимодействующих процессов с большим количеством соединений между ними. Соответственно, для каждого со-

единения появляется своя точка синхронизации, в которой процесс-читатель может узнать, что очередь не пуста.

Необходимо разработать метод оповещения процесса-читателя о появлении данных в разделяемой памяти, которые ему необходимо принять и обработать.

2.4. Методы оповещения о появлении данных в разделяемой памяти

2.4.1. Наивные алгоритмы в разделяемой памяти

Процесс-читатель знает о расположении всех очередей в разделяемой памяти, в которые отправляют сообщения все процессы-писатели, с которыми он взаимодействует. Непосредственно само состояние очередей может быть использовано для оповещения процесса-читателя о наличии данных в этих очередях.

Алгоритм №1 При небольшом количестве соединений (например, $0.25 * N$, где N - количество ядер в процессоре) возможно использование выделенных потоков в процессе-читателе для активного опроса состояния очереди и обработки соответствующего соединения.

Алгоритм №2 При большем количестве соединений возможно использовать группу выделенных потоков, активно опрашивающих которое количество очередей в разделяемой памяти. Например, 1 поток, активно опрашивающий до 10 соединений.

2.4.1.1. Применимость, достоинства и недостатки

Данные алгоритмы активного опроса очередей в разделяемой памяти для обслуживания соединений вполне могут быть использованы в реальных системах. Их можно применить в системах: с большими вычислительными ресурсами, небольшим количеством процессов и очень активных соединений.

В противном случае, активно работающие потоки будут выполнять много бесполезных операций по опросу пустых очередей неактивных соединений. Кроме того, постоянно работающий поток, опрашивающий очереди в разделяемой памяти, может быть вытеснен с процессора планировщиком операционной системы из-за израсходования отведенного ему кванта процессорного времени, что ухудшит качество обслуживания заявок.

Приведенные методы в настоящей работе не рассматриваются, поскольку количество соединений между процессами на одном физическом узле и самих процессов может быть большим, а опрос очередей на наличие в них данных сопровождается взятием взаимной блокировки, что приводит к неэффективному использованию аппаратных ресурсов.

2.4.2. TCP

Как было сказано выше, TCP используется как базовый метод межпроцессного взаимодействия. Он может быть использован и как метод оповещения о появлении данных в очереди в разделяемой памяти.

2.4.2.1. Алгоритм взаимодействия при использовании TCP для оповещения о появлении данных

а) процессу-читатель ожидает новых данных по всем своим TCP соединениям в состоянии сна в системном мультиплексоре событий;
б) для передачи данных таким методом процессу-писателю необходимо записать в очередь нужное сообщение и передать на нижележащий модуль сообщение минимального размера в 1 байт с заранее установленным значением (например, ”0“);
в) ядро операционной системы пробуждает процесс-читатель;
г) реактор процесса-читателя демультиплексирует активное TCP соединение, считывает 1 байт полезных данных и отправляет его на следующий слой обработки межпроцессных взаимодействий через ранее описанный стек модулей;
д) модуль, отвечающий за взаимодействие по разделяемой памяти, проверяет, что полученный 1 байт имеет заранее оговоренное значение (”0“ в примере выше) и это служит для него сигналом к проверке состояния очереди в разделяемой памяти для соединения, с которого этот сигнальный байт был получен;
е) процесс-читатель считывает сообщение из очереди и выполняет его обработку.

Таким образом происходит передача данных между процессами в разделяемой памяти с оповещением о появлении данных в разделяемой памяти по TCP.

2.4.2.2. Работа с очередью в разделяемой памяти при использовании TCP для оповещения о появлении данных

2.4.2.3. Достоинства и недостатки

Предложенный метод обладает следующими **достоинствами**:

— позволяет эффективно поддерживать множество соединений с использованием системных мультиплексоров событий (*select/poll/epoll*);
— позволяет процессу-читателю блокировать свое выполнение до появления оповещения;
— благодаря использованию эвристики из раздела 2.4.2.2 временная задержка на передачу данных может быть снижена, если к концу обработки очередного сообщения в очередь уже будет записано новое сообщение;
— метод межпроцессного взаимодействия по TCP не требует доработки и может быть использован как есть.

Недостаток у такого подхода только один: временная задержка на отправку и получение оповещения. Основные отличия от метода, использующего только TCP, в том, как передается само сообщения. Но кроме использования среды для передачи самих данных выполняется рад системных вызовов для оповещения процесса-читателя, каждый из которых может вносить существенную временную задержку:

— *write* – для записи 1 байта в TCP-сокет процессом-писателем;
— *select/poll/epoll* – для демультиплексирования нужного события среди множества источников процессом-читателем.
— *read* – для чтения 1 байта из TCP-сокета процессом-читателем.

В случае, когда процесс-читатель находится в состоянии сна на системном мультиплексоре событий, процесс-писатель в ходе системного вызова *write* должен также изменить состояние процесса-читателя на ”Готов к выполнению“ или ”Выполняется“. После этого в течение некоторого промежутка времени процесс-читатель будет готовиться к выполнению. Все это может влиять на временную задержку на передачу данных.

Следовательно, необходимо разработать новый метод мультиплексирования соединений, использующих разделяемую память, имеющий меньшие накладные расходы на использование и обладающий, как минимум, теми же достоинствами, что и описанный в данном разделе.

2.4.3. Мультиплексор событий в разделяемой памяти

С целью избежать излишних накладных расходов и использовать системные ресурсы наилучшим образом в настоящей работе предлагается метод мультиплексирования событий от множества соединений, использующих разделяемую память для передачи данных.

Мультиплексор событий в разделяемой памяти – это структура данных, используемая множеством процессов-писателей для оповещения процесса-читателя о появлении данных в очереди в разделяемой памяти. Каждый процесс, участвующий в межпроцессном взаимодействии, должен иметь свой мультиплексор событий, через который ему будут поступать оповещения о появлении данных в разделяемой памяти, которые он может считать и обработать.

Время жизни мультиплексора событий определяется процессом-читателем. При необходимости процесс-читатель создает файл определенного размера в ФС и отображает его в свою память. Во время установления соединения процесс-читатель ассоциирует соединение с номером от 0 до 2047 и отправляет его вместе с путем до файла другой стороне. Эти данные используются противоположной стороной для отправки оповещений.

2.4.3.1. Структура и алгоритм работы мультиплексора событий в разделяемой памяти

Структура мультиплексора событий представлена на Рисунке 1. Он состоит из 4-байтного целого числа futex, используемого для синхронизации взаимодействующих процессов, и массив из 32 8-байтных сигнальных чисел, по одному на каждый бит futex. Эти 32 8-байтных числа содержат 2048 бит, что позволяет различать 2048 различных соединений. Описание на языке C++ представлено на Листинге 3.

Когда процесс-писатель хочет оповестить процесс-читатель о наличии данных в очереди в разделяемой памяти, ему необходимо:

а) атомарно выставить бит в сигнальном числе, соответствующий этому соединению;
б) атомарно выставить соответствующий бит futex и получить его предыдущее значение;
в) Если предыдущее значение равно нулю, значит, разбудить процесс-читатель, чтобы он мог обработать оповещение.

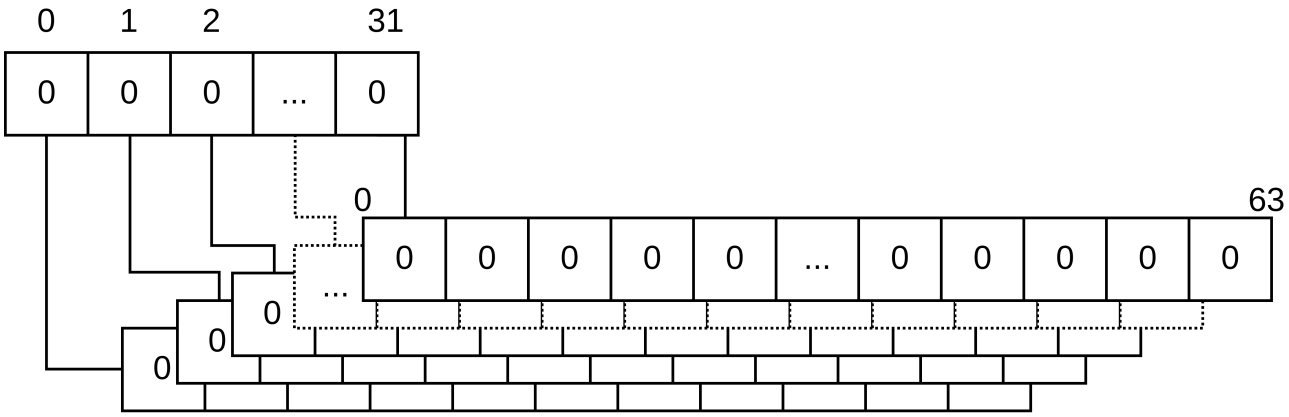

Рисунок 1 – Структура мультиплексора событий в разделяемой памяти

г) Если оно не равно нулю, это значит, что другой процесс-писатель уже либо разбудил целевой процесс-читатель, либо в скором времени сделает это (так как он был тем самым процессом, перед которым в futex был нуль).

После завершения первых двух этапов мультиплексор для соединения под номером #1987 будет находиться в состоянии, представленном на рисунке 2. Чтобы проставить нужные биты для данного, процесс-писатель делит номер его соединения на 64 и получает индекс бита в futex – 31, и, соответственно, индекс сигнального числа. Далее он вычисляет остаток от деления номера сигнала на 64 и получает индекс бита в сигнальном числе. После чего атомарной операцией ”ИЛИ“ выставляется сначала нужный бит в сигнальном числе, а потом нужный бит в futex. Псевдокод алгоритмов процесса-писателя и читателя приведены на Листингах А.2 и А.1.

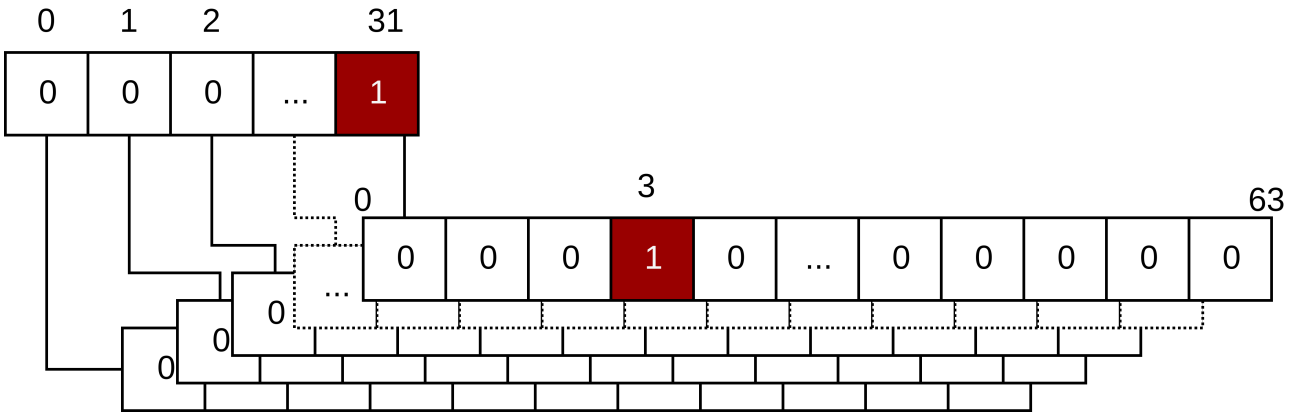

Рисунок 2 – Состояние мультиплексора событий в разделяемой памяти с активным сигналом #1987

Листинг 3 – Структура мультиплексора в памяти

```
struct Multiplexer
{
    using Futex = std::atomic<int32_t>;
    using Signal = uint64_t;

    static constexpr std::size_t c_num_chunks = sizeof(Futex) *
   CHAR_BIT;
    static constexpr std::size_t c_signals_per_chunk = sizeof(std::
   atomic<Signal>) * CHAR_BIT;

    // Процедура ожидания на futex
    void wait() {
        if (!m_futex) {
            futex_wait(&m_futex, 0);
        }
    }

    // Процедура оповещения процессачитателя-. Выставляет
   соответствующие биты мультиплексора для сигнала за номером signal
   и при необходимости пробуждает поток мультиплексора
   процессачитателя-.
    void notify(Multiplexer::Signal id);

    // Процедура для пробуждения потока, спящего на futex.
    void wakeup();

protected:
    // байтное4- число futex, на котором происходит синхронизация
   снапробуждения/ потока мультиплексора.
    Futex m_futex;

    // Для избежания лишней состязательности между атомарными
   операциями над массивом сигнальных чисел и над futex, массив
   выровнен на размер кэшлинии- процессора -- 64 байта.
    alignas(64) std::array<std::atomic<Signal>, c_num_chunks>
   m_signals;
};
```

2.4.3.2. Достоинства и недостатки

Данный метод решает проблему, описанную в Разделе 2.4.1.1, позволяя определять соединение-источник сигнала за время, не зависящее от количества соединений.

По сравнению с вышеописанным методом межпроцессного взаимодействия, использующим TCP для оповещения о появлении данных в разделяемой памяти, данное решение обладает следующими **достоинствами**:

а) Подавляющая часть работы происходит в пользовательском пространстве. В лучшем сценарии отправка и получение оповещения не задействуют системные вызовы. Оповещение происходит посредством двух атомарных операций.
б) Ядро ОС используется только пробуждения и засыпания процессов.

Таким образом, в худшем случае совершается один системный вызов для засыпания процесса в ожидании оповещений и один для пробуждения процесса. Пробуждение потока влечет временные затраты как со стороны процесса-писателя – на пробуждение потока, так и со стороны процесса-читателя – на уход в состояние сна и временную задержку на постановку потока мультиплексора на выполнение. В то же время, ожидание оповещений в состоянии сна позволяет экономить ресурс процессора.

Недостатком данного метода является механизм управление файлом мультиплексора событий в файловой системе. Поскольку файлом управляет процесс, то в случае его некорректного завершения файл не будет удален и останется на всегда. Некорректное завершение может произойти по следующим причинам: крах процесса вследствие программного дефекта, при отправке сигнала *SIGKILL* процессу, который в большинстве случаев приводит к немедленному завершению процесса.

Данный недостаток можно решить, используя более продвинутый механизм создания файлов в ФС. Например, делегировать эту работу отдельному процессу или группе процессов. В таком случае такой процесс может отслеживать состояние своих клиентских процессов и при прекращении их работы выполнять подчистку ресурсов.

2.4.4. Методы обработки оповещений в мультиплексоре событий

Имея более совершенный механизм оповещения процессов о появлении данных в разделяемой памяти необходимо разработать также и метод обработки получаемых оповещений. В данном подразделе предложены четыре метода:

а) Синхронный – существует единственный выделенный поток мультиплексора. Он занимается непосредственно обслуживанием соединений.
б) ”Полусинхронный/Полуреактивный“ – существует единственный выделенный поток мультиплексора (полуреактивная часть). Он диспетчеризует синхронную обработку оповещений в пул потоков (полусинхронная часть) [13].

в) ”Лидер/Последователи“ – в один момент времени только один поток-лидер отслеживает состояние мультиплексора в пассивном режиме. То есть, при отсутствии оповещений процесс-лидер находится в состоянии сна. При обнаружении оповещений он передает лидерство произвольному потоку из пула и переходит к обработке оповещений [8].

г) ”Лидер/Последователи“ – в один момент времени только один поток-лидер активно отслеживает состояние мультиплексора. То есть, постоянно опрашивает мультиплексор. При обнаружении оповещений он передает лидерство произвольному потоку из пула и переходит к обработке оповещений.

2.4.4.1. Синхронный метод обработки соединений

2.4.4.2. Метод обработки соединений ”Полусинхронный/Полуреактивный“

Представлен в ранней работе автора [1].

2.4.4.3. Метод обработки соединений ”Лидер/Последователи“

2.4.4.4. Метод обработки соединений ”Лидер/Последователи“ с активным ожиданием оповещений

Выводы по главе 2

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ И ОБРАБОТКА РЕЗУЛЬТАТОВ

3.1. Постановка эксперимента

3.1.1. Конфигурация экспериментального стенда

3.1.1.1. Аппаратное обеспечение

Процессоры 2 x Intel Xeon Platinum 8168 CPU @ 2.7 GHz. Технология Hyper-Threading отключена для уменьшения нежелательного влияния на время обслуживания заявок в системе [7].

Оперативная память: DDR4-2666 128 GiB.

3.1.1.2. Программное обеспечение

Операционная система Red Hat Enterprise Linux Server release 7.8 (Maipo). Ядро Linux 3.10.0-1127.el7.x86_64

Компилятор C++ Clang 6.0.1.

Стандартная библиотека C++: libstdc++ 8.1.0.

Библиотека Boost.Interprocess 1.68.0.

3.1.2. Конфигурация экспериментальной системы

Система для проведения эксперимента состоит из двух процессов:

— Процесс-шлюз отвечает за преобразование заявок из формата внешнего мира во внутренний формат системы и обратно.
— Процесс-обработчик совершает некоторые преобразования над заявкой и отправляет результат за пределы системы через процесс-шлюз.

Процессы выполняются на двух процессорах, расположенных в разных разъемах на материнской плате физического узла.

Снаружи системы находится симулятор внешнего мира. Он генерирует поток заявок в систему и получает результат обработки заявки в системе. Схема взаимодействия процессов в эксперименте представлена на Рисунке 3.

В настоящей работе замеряется временная задержка на передачу данных между процессами внутри системы, а именно из процесса-шлюза в процесс-обработчик и обратно (сообщения типа №1 и №2 в запросе и ответе между процессом-шлюзом и процессом обработчиком на Рисунке 3).

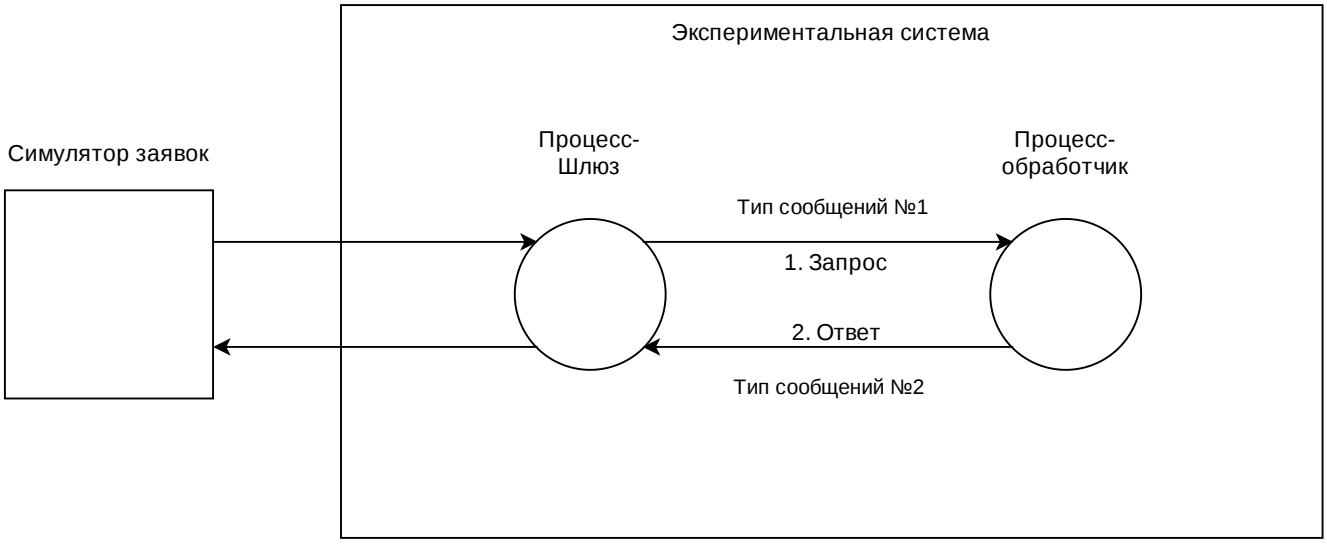

Рисунок 3 – Схема взаимодействия процессов в эксперименте

Процессы системы взаимодействуют используют одно соединение, в рамках которого заявки обрабатываются строго последовательно. Обслуживание включает в себя: прием заявки, выполнение пользовательской логики над заявкой и, если необходимо, отправка ответа. Временной задержкой на передачу данных в настоящей работе принимается временной промежуток от начала отправки заявки до **начала обработки заявки**. Таким образом, возможен случай, когда во время обработки очередной заявки процессом в очереди уже находится следующая заявка, временная задержка на передачу которой, таким образом, увеличится на время обработки текущей заявки.

Данный сценарий актуален для процесса-обработчика, в котором обслуживание заявки осуществляется непосредственно в транспортном потоке В случае с процессом-шлюзом транспортный поток только читает и диспетчеризует асинхронную обработку заявки, т.е. не выполняет обработку самой заявки.

Пользовательская логика процесс-обработчика в среднем отклоняет 25% заявок, в то время как 75% заявок отправляются в процесс-шлюз.

3.1.3. Используемые обозначения

Все величины с, указанные с интервалом, указаны с 95% доверительной вероятностью.

- Δ – временная задержка между сериями заявок;
- δ – временная задержка между заявками в серии;
- транспортный поток – поток из пула транспортных потоков, непосредственно обслуживающий получаемые заявки.

3.1.4. Характер экспериментальной нагрузки

Симулятор отправляет в систему заявки сериями с интервалом Δ в *10 мс* по *4* заявки в серии с интервалом δ *60 мкс* между ними. Так как для каждого эксперимента производился отдельный запуск симулятора, средние значения обозначенных величин приведены для каждого эксперимента отдельно.

3.1.5. Время обслуживания заявок в процессах

На Рисунке 4 представлена гистограмма времени обслуживания заявок в процессе-обработчике. Время обслуживания заявок с 95% доверительной вероятностью укладывается в диапазон 13 ± 7 *мкс*.

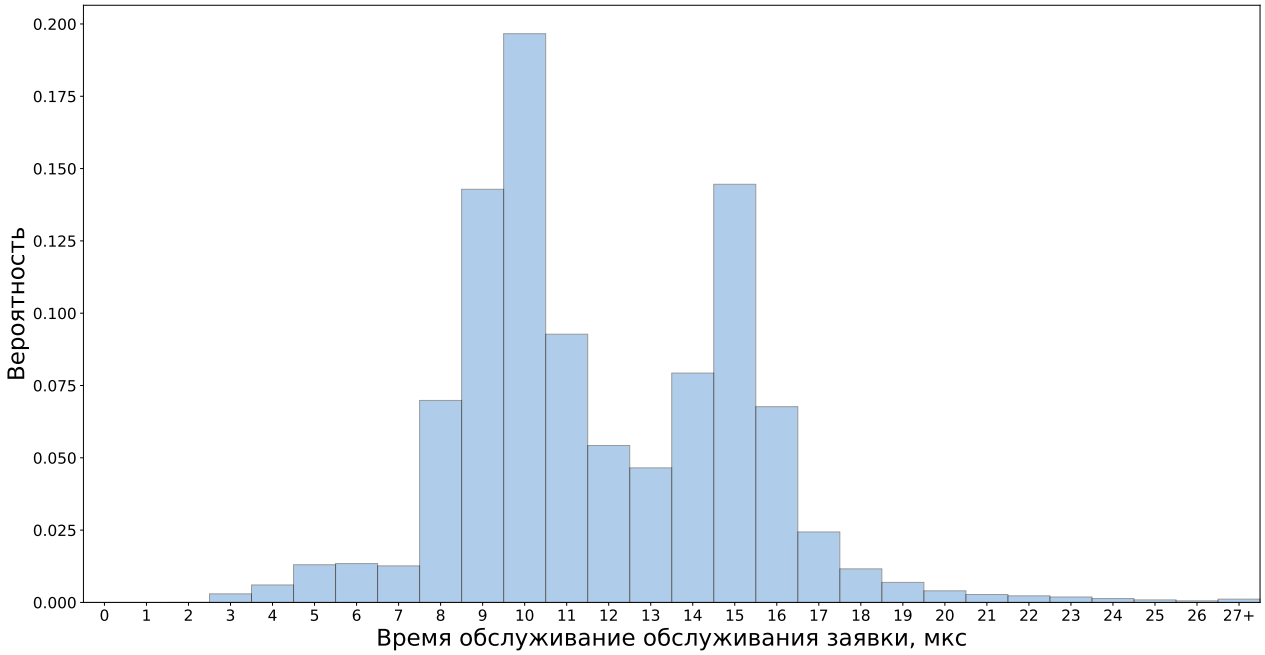

Рисунок 4 – Гистограмма времени обслуживания заявки в процессе-обработчике

На Рисунке 5 представлена гистограмма времени обслуживания заявок в процессе-шлюзе. Время обслуживания заявок с 95% доверительной вероятностью укладывается в диапазон 19 ± 6 *мкс*.

Как было сказано выше, в процессе-шлюзе заявки обслуживания вне транспортного потока, поэтому время обслуживания заявок в процессе-шлюзе не влияет на временную задержку на передачу данных. В случае с процессом-обработчиком значительная часть обслуживания заявки выполняется именно в транспортном потоке, что влияет на временную задержку на передачу данных, т.к. нахождение в очереди влияет на данный показатель.

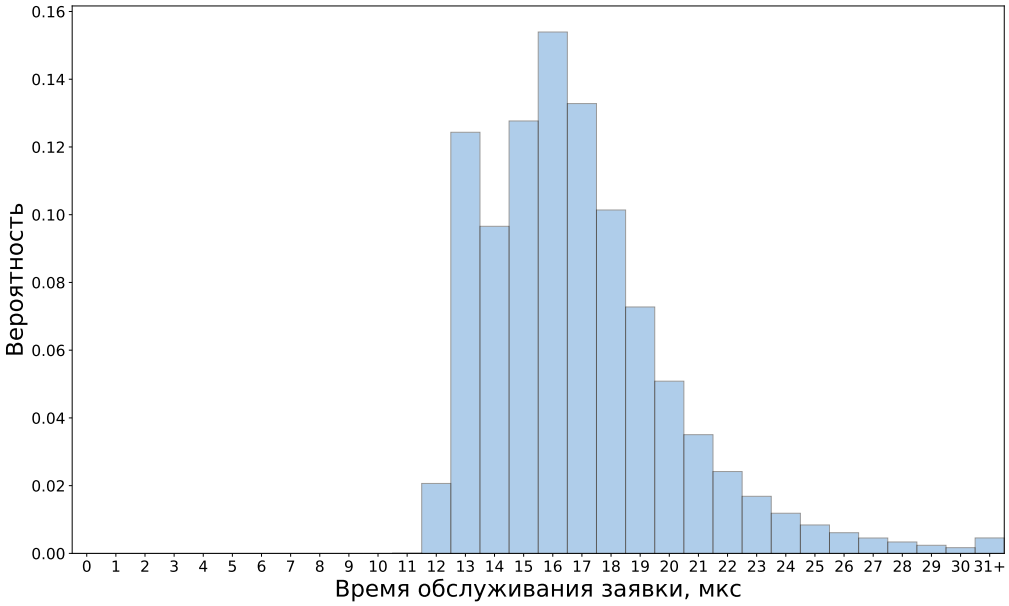

Рисунок 5 – Гистограмма времени обслуживания заявки в процессе-шлюзе

3.2. Использование TCP для передачи данных

В качестве точки отсчета в настоящей работе выступает метод межпроцессного взаимодействия на основе TCP, используемый посредством сокетов 2.2.

Показатели потока исходящих заявок из симулятора в данном эксперименте: $\Delta \in 10 \pm 4$ *мс*, $\delta \in 50 \pm 26$ *мкс*.

Гистограмма временной задержки на передачу данных для данного метода приведена на Рисунке 6. В Таблице 1 приведены основные временные характеристики данного метода.

Таблица 1 – Основные показатели временной задержки на передачу данных для метода на основе TCP

Направление взаимодействия/ Показатель	Процесс-шлюз → Процесс-обработчик	Процесс-обработчик → Процесс-шлюз
min(t), мкс	9	13
M(t), мкс	27.5 ± 8.5	28 ± 7
max(t), мс	2.1	9.2
Δ, мс	10 ± 4	10 ± 4
δ, мкс	50 ± 26	87 ± 32

Временная задержка на передачу данных в обоих направлениях имеет схожие значения. Это объясняется тем, что накладные расходы на использование TCP

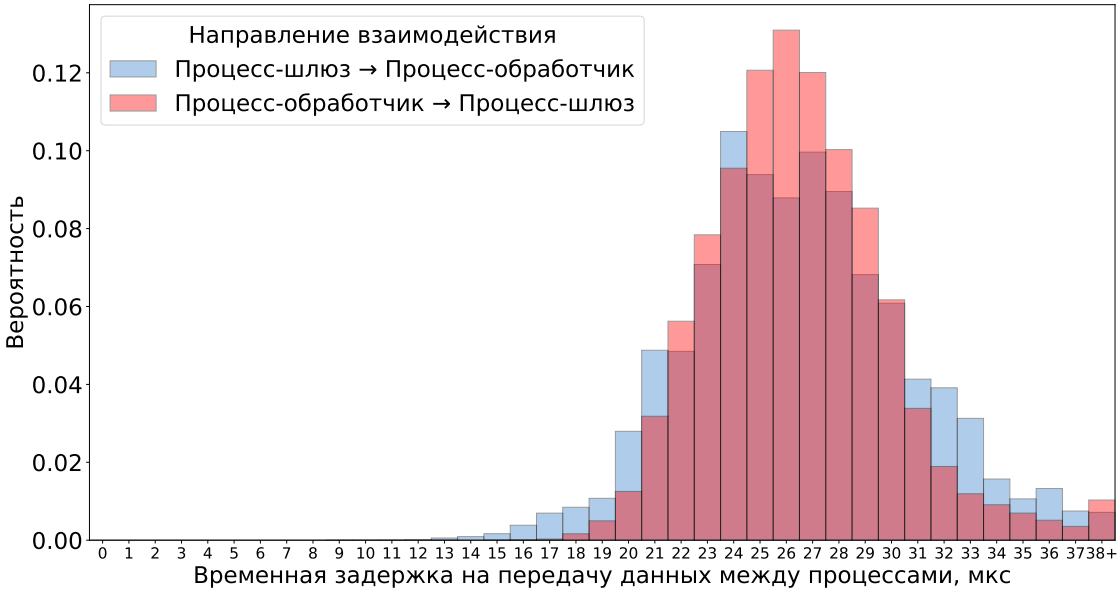

Рисунок 6 – Гистограмма временной задержки на передачу данных между процессами при использовании TCP

через механизм сокетов имеют подавляющее значение над прочими факторами, влияющими на межпроцессное взаимодействие.

3.3. Использование разделяемой памяти для передачи данных

3.3.1. Использование TCP для оповещения о появлении данных

Показатели потока исходящих заявок из симулятора в данном эксперименте: $\Delta \in 10 \pm 4$ *мс*, $\delta \in 50 \pm 27$ *мкс*.

В данном подразделе приведены данные об экспериментах с методом межпроцессного взаимодействия, описанным в Разделе 2.4.2.

В Таблице 2 приведены основные временные характеристики данного метода. На Рисунке 7 приведена гистограмма временной задержки на передачу данных для данного метода.

Как описано выше, значительная часть обслуживания заявки процессом-обработчиком происходит непосредственно в транспортном потоке. Из-за этого к моменту конца обработки текущей заявки очередная заявка уже может находиться в очереди в разделяемой памяти, что позволяет использовать оптимизацию, описанную в Разделе 2.4.2.2. А именно принять и начать обслуживание очередной заявки, не используя дорогостоящий механизм оповещения, если заявка уже находится в очереди в разделяемой памяти.

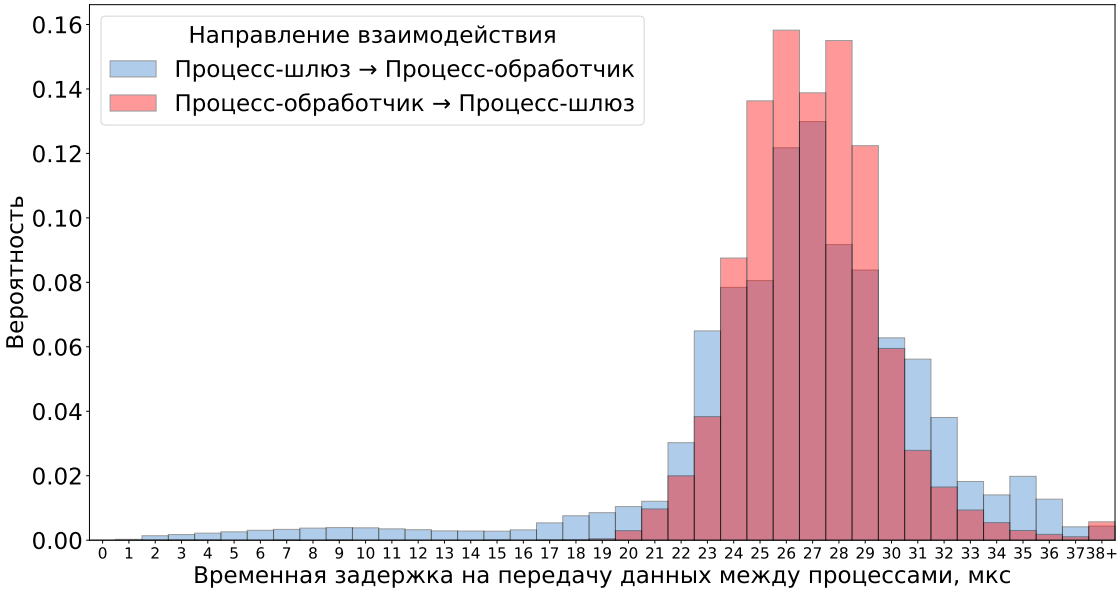

Рисунок 7 – Гистограмма временной задержки на передачу данных между процессами при использовании разделяемой памяти для передачи данных и TCP для оповещения о появлении данных в ней

Таблица 2 – Основные показатели временной задержки на передачу данных для метода, использующего разделяемую памяти для передачи данных и TCP для оповещения о появлении данных в ней

Направление взаимодействия/ Показатель	Процесс-шлюз → Процесс-обработчик	Процесс-обработчик → Процесс-шлюз
min(t), мкс	1	3
M(t), мкс	22.5 ± 12.5	27.5 ± 5.5
max(t), мс	3	9.8
Δ, мс	10 ± 4	10 ± 4
δ, мкс	50 ± 27	87 ± 32

Прием заявки процессом-шлюзом не связан с обслуживанием заявки, поэтому к моменту, когда процесс-шлюз заканчивает прием и диспетчеризацию заявки, очередь входящих заявок в данном эксперименте пуста и поток-шлюз переходит к пассивному ожиданию новых заявок. Таким образом, приему и обслуживанию большинства заявок сопутствует пассивное ожидание сигнала по TCP, что негативно сказывается на временной задержке на передачу данных.

3.3.2. Использование мультиплексора в разделяемой памяти для оповещения о появлении данных

В данном подразделе приведены данные об экспериментах с семейством методов межпроцессного взаимодействия, описанными в Разделе 2.4.3.

3.3.2.1. Методы с пассивным ожиданием оповещений

В методах с пассивным ожиданием оповещений поток мультиплексора событий использует примитив *futex* для ожидания новых сигналов (см. Разделы 2.4.4.2 и 2.4.4.3). Поток процесса-читателя, опрашивающий мультиплексор в разделяемой памяти, находится в состоянии сна и пробуждается процессом-писателем при необходимости.

Диспетчеризация и обработка соединений по модели "Полусинхронный/Полуреактивный" Показатели потока исходящих заявок из симулятора в данном эксперименте: $\Delta \in 10 \pm 4$ *мс*, $\delta \in 51 \pm 28$ *мкс*.

В данном подразделе приведены данные об экспериментах с методом межпроцессного взаимодействия, описанным в Разделе 2.4.4.2.

Таблица 3 – Основные показатели временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, пассивное ожидание оповещений в мультиплексоре событий в разделяемой памяти и метод "Полусинхронный/Полуреактивный" при обслуживании заявок

Направление взаимодействия/ Показатель	Процесс-шлюз → Процесс-обработчик	Процесс-обработчик → Процесс-шлюз
min(t), мкс	1	3
M(t) $\pm$ 95%, мкс	12.5 $\pm$ 5.5	11.5 $\pm$ 2.5
max(t), мс	6.9	11.6
Δ, мс	10 $\pm$ 4	10 $\pm$ 4
δ, мкс	51 $\pm$ 28	92 $\pm$ 36

В Таблице 3 приведены основные временные характеристики данного метода. На Рисунке 8 приведена гистограмма временной задержки на передачу данных для данного метода.

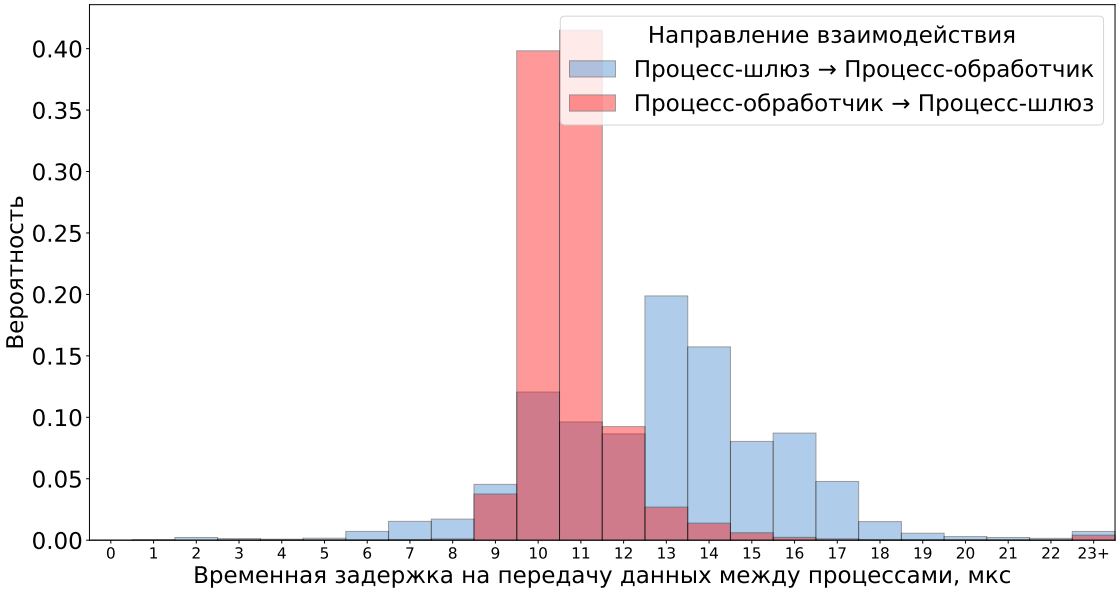

Рисунок 8 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, пассивное ожидание событий на мультиплексоре событий в разделяемой памяти и метод ”Полусинхронный/Полуреактивный“ при обслуживании заявок

Использовании более эффективного метода межпроцессного взаимодействия наглядно показывает эффект от влияния времени обслуживания заявки в транспортном потоке на временную задержку на передачу данных. Процесс-шлюз с минимальной временной задержкой принимает и диспетчеризует заявку для дальнейшей обработки и сразу же готов принимать следующую заявку. Процесс-обработчик же использует транспортный поток для частичного обслуживания заявки (см. Рисунок 4), а потому в среднем временная задержка на передачу данных имеет худшие показатели, так как это может задерживать прием очередных заявок.

Диспетчеризация и обработка соединений по модели ”Лидер/Последователи”

Показатели потока исходящих заявок из симулятора в данном эксперименте: $\Delta \in 10 \pm 4$ *мс*, $\delta \in 51 \pm 28$ *мкс*.

В данном подразделе приведены данные об экспериментах с методом межпроцессного взаимодействия, описанным в Разделе 2.4.4.3.

В Таблице 4 приведены основные временные характеристики данного метода.

На Рисунке 9 приведена гистограмма временной задержки на передачу данных для данного метода.

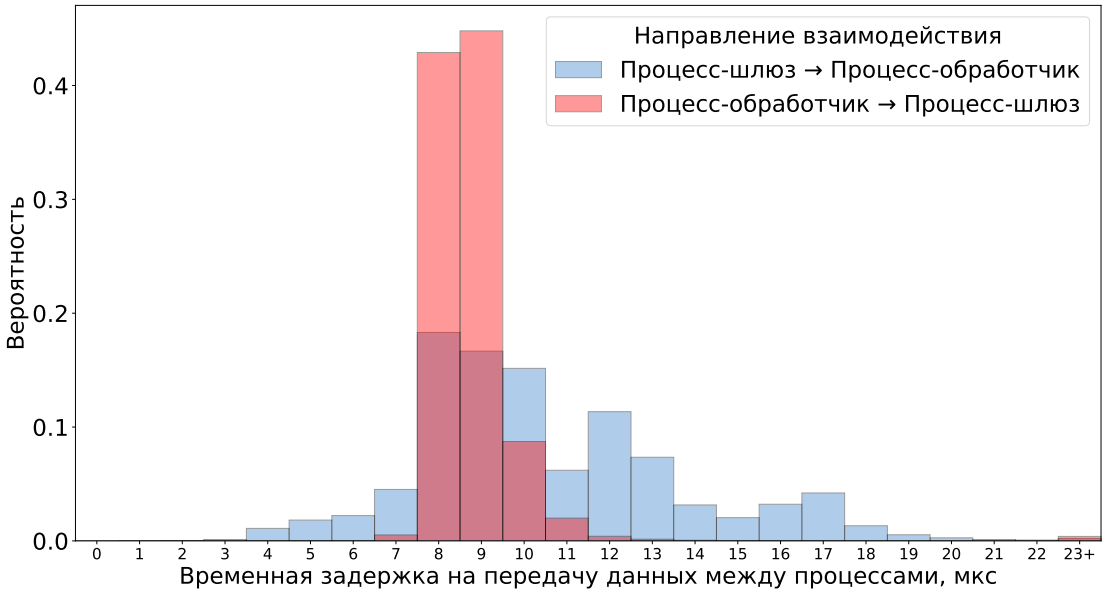

Рисунок 9 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, пассивное ожидание оповещений в мультиплексоре событий в разделяемой памяти и метод ”Лидер/Последователи“ при обслуживании заявок

Таблица 4 – Основные показатели временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, пассивное ожидание оповещений в мультиплексоре событий в разделяемой памяти и метод ”Лидер/Последователи“ при обслуживании заявок

Направление взаимодействия/ Показатель	Процесс-шлюз → Процесс-обработчик	Процесс-обработчик → Процесс-шлюз
min(t), мкс	1	2
M(t) $\pm$ 95%, мкс	11.5 $\pm$ 6.5	9.5 $\pm$ 1.5
max(t), мс	2.4	9.5
Δ, мс	10 $\pm$ 4	10 $\pm$ 4
δ, мкс	51 $\pm$ 28	89 $\pm$ 33

По сравнению с методом ”Полусинхронный/Полуреактивный“ данный метод ожидаемо имеет меньшую временную задержку на передачу данных. В данном методе поток, получивший оповещение из мультиплексора событий, приступит к обработке сразу после того, как пробудит следующий поток-лидер. В то

время как для предыдущего метода обработка заявки начнется только после пробуждения отдельного потока.

Выводы по исследованиям методов с пассивным ожиданием оповещений Методы оповещения процессов о появлении данных в разделяемой памяти с использованием мультиплексора в разделяемой памяти показывают существенно меньшую временную задержку на доставку оповещения, чем метод с использованием TCP. Это объясняется существенно меньшим количеством системных вызовов для обеих сторон взаимодействия.

Полученный результат подтверждает тезисы автора этих методов о превосходстве метода ”Лидер/Последователи“ над методом ”Полусинхронный/Полуреактивный“ при отсутствии необходимости приоритезации обработки заявок [11, с. 398]. Однако, в отличие от исходного метода ”Полусинхронный/Полуреактивный“ [11, с. 375], работающего с сокетами и системным мультиплексором событий, в текущей ситуации нет необходимости считывать данные из сокета, чтобы переложить их в централизованную очередь для последующей обработки. Поэтому преимущество не такое существенное.

3.3.2.2. Метод с активным ожиданием оповещений

В методе с активным ожиданием оповещений поток мультиплексора событий находится в режиме постоянного опроса мультиплексора на предмет соединений (см. Раздел 2.4.4.4). В данном параграфе рассматривается исключительно метод обслуживания заявок ”Лидер/Последователи“ т.к. в параграфе выше он показал лучший результат по сравнению с методом обслуживания заявок ”Полусинхронный/Полуреактивный“.

Диспетчеризация и обработка соединений по модели ”Лидер/Последователи” Показатели потока исходящих заявок из симулятора в данном эксперименте: $\Delta \in 10 \pm 4$ *мс*, $\delta \in 51 \pm 27$ *мкс*.

В данном подразделе приведены данные об экспериментах с методом межпроцессного взаимодействия, описанным в Разделе 2.4.4.4.

В Таблице 5 приведены основные временные характеристики данного метода. На Рисунке 10 приведена гистограмма временной задержки на передачу данных для данного метода.

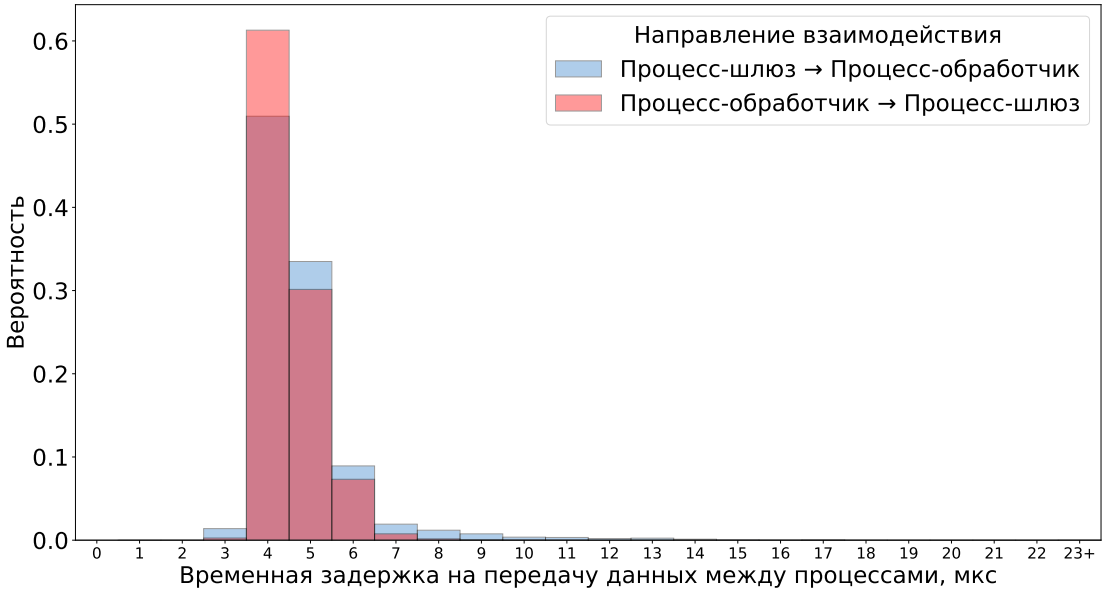

Рисунок 10 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, активно опрашиваемый мультиплексор в разделяемой памяти и модель ”Лидер/Последователи“ при обслуживании заявок

Таблица 5 – Основные показатели временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, активно опрашиваемый мультиплексор в разделяемой памяти и модель ”Лидер/Последователи“ при обслуживании заявок

Направление взаимодействия/ Показатель	Процесс-шлюз → Процесс-обработчик	Процесс-обработчик → Процесс-шлюз
min(t), мкс	1	3
M(t) $\pm$ 95%, мкс	6 ± 2	5 ± 1
max(t), мс	0.064	0.017
Δ, мс	10 ± 4	10 ± 4
δ, мкс	51 ± 28	87 ± 33

Выводы по исследованию метода с активным ожиданием Метод с активным ожиданием событий в мультиплексоре в разделяемой памяти ожидаемо показывает лучший результат по сравнению со всеми ранее рассмотренными методами. Это происходит потому что в данном случае нет необходимости пробуждать и дожидаться пробуждения потока для обслуживания заявки. Исходя из разницы временной задержки на передачу данных в методах с активным и пассив-

ным ожиданием, пробуждение потока до постановки его на выполнение занимает около 5-6 мкс. В работе другого автора [6] была измерена временная задержка на пробуждение потоков, прикрепленных к разным ядрам разных процессоров, посредством системного вызова futex. Измерения проводились на процессоре AMD Opteron 6272 и показали $\mu = 24640.5$ машинных циклов от системного вызова до выполнения первой инструкции пробужденным потоком. Или $\frac{24640.5 \text{ машинных циклов}}{2.1 * 10^9 \text{ Гц}} \approx 11$ мкс при частоте процессора 2.1 ГГц, что похоже на обозначенный выше результат с учетом использования в настоящей работе более современного аппаратного обеспечения.

Для данного метода распределение временной задержки на передачу данных от процесса-обработчика к процессу-шлюзу схоже с предыдущими методами, но этот показатель существенно отличается при передаче от процесса-шлюза к процессу-обработчику. Это может быть вызвано тем, что более быстрый метод межпроцессного взаимодействия при данной постановке эксперимента приводит к отсутствию и, как следствие, время обслуживания заявки в транспортном потоке процесса-обработчика не влияет на временную задержку на передачу данных.

Данный подход обладает существенным **недостатком**. Поскольку поток, активно опрашивающий мультиплексор, исполняется до получения и обработки события, то существует возможность, что планировщик ОС вытеснит этот поток с процессора. Стандартный квант планирования потоков в Linux – 100 мс. Если событие не будет получено и обработано за это время, то поток может быть вытеснен с процессора и временная задержка на передачу данных увеличится на сотни миллисекунд. Данный недостаток не проявляется в проведенном эксперименте, поскольку что интервал Δ между сериями заявок значительно меньше 100 мс.

Выводы по главе 3

Проведено экспериментальное сравнение разработанных методов межпроцессного взаимодействия.

а) Методы межпроцессного взаимодействия, использующие мультиплексор в разделяемой памяти для оповещения о появлении данных в очереди в разделяемой памяти имеют существенно меньшую временную задержку на передачу данных, чем метод, использующий для этого TCP. А именно, 11.5 ± 6.5 мкс и 9.5 ± 1.5 мкс для пассивного варианта

”Лидер/Последователи“ против 22.5 ± 12.5 мкс и 27.5 ± 5.5 мкс для TCP с передачей данных через очередь в разделяемой памяти.

б) В семействе пассивных методов межпроцессного взаимодействия на основе мультиплексора в разделяемой памяти наименьшую временную задержку на передачу данных показала вариация с использованием метода обслуживания заявок ”Лидер/Последователи“, а именно 11.5 ± 6.5 мкс и 9.5 ± 1.5 мкс против 12.5 ± 5.5 мкс и 11.5 ± 2.5 мкс.

в) Самой низкой временной задержки на передачу данных удалось добиться при использовании активно опрашивающей мультиплексор вариации метода межпроцессного взаимодействия, использующего метод ”Лидер/Последователи“ при обслуживании заявок. А именно, 6 ± 2 мкс и 5 ± 1 мкс.

г) При использовании пассивных методов на основе мультиплексора в разделяемой памяти заметны эффекты от обслуживания заявок в транспортном потоке на временную задержку на передачу данных. В проведенном эксперименте в процессе-шлюзе заявки частично обслуживаются именно в транспортном потоке, из-за чего могут образовываться очереди и увеличиваться временная задержка на передачу данных. В активно опрашивающем методе этот эффект не заметен, так как временная задержка на передачу данных меньше, соответственно, меньше временная задержка реакции на очередную заявку и, следовательно, меньше возможностей для формирования очереди из заявок.

д) Активно опрашивающий мультиплексор метод обладает существенным недостатком. поток, активно опрашивающий мультиплексор в разделяемой памяти, может быть вытеснен с процессора по окончании отведенного ему кванта процессорного времени – 100 миллисекунд. В проведенном эксперименте данный эффект не наблюдается, так как серии заявок отправляются симулятором в систему на порядок чаще, каждые 10 ± 4 миллисекунд. Данный недостаток может быть необходимо разрешить для обеспечения должного уровня качества обслуживания заявок в системе.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Губарев В. Ю.* Реализация методов эффективного взаимодействия процессов в распределенных системах // Сборник тезисов докладов конгресса молодых ученых. Электронное издание. — Университет ИТМО, 2020.

2 *Кузичкина А. О.* Исследование и разработка методов трассировки событий в параллельных и распределенных системах : Дипломная работа / Кузичкина Анастасия Олеговна. — Факультет ПИиКТ : Университет ИТМО, 2017.

3 Comparing and Evaluating epoll, select, and poll Event Mechanisms / L. Gammo [et al.] // Proceedings of the 6th Annual Ottawa Linux Symposium. — 2004.

4 Draft: Have you checked your IPC performance lately? [Электронный ресурс] / S. Smith [et al.]. — 2012. — URL: https://www.semanticscholar.org/paper/Draft-%3A-Have-you-checked-your-IPC-performance-Smith-Madhavapeddy/1cb3b82e2ef6a95b576573b8af0f3ec6f7bc21d9 (дата обращения: 25.03.2020).

5 *Gaztanaga I.* Chapter 18. Boost.Interprocess [Электронный ресурс]. — URL: https://www.boost.org/doc/libs/1_63_0/doc/html/interprocess.html (дата обращения: 25.03.2020).

6 *Hale K. C., Dinda P. A.* An Evaluation of Asynchronous Software Events on Modern Hardware // 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). — 2018. — P. 355–368.

7 *Khartchenko E.* Optimizing Computer Applications for Latency: Part 1: Configuring the Hardware [Электронный ресурс]. — 2017. — URL: https://software.intel.com/content/www/us/en/develop/articles/optimizing-computer-applications-for-latency-part-1-configuring-the-hardware.html (дата обращения: 28.03.2020).

8 Leader/followers / D. C. Schmidt [et al.] // Proceeding of the 7th Pattern Languages of Programs Conference. — 2000. — P. 1–40.

9 LTTng: an open source tracing framework for Linux [Электронный ресурс]. — URL: https://lttng.org/ (дата обращения: 25.03.2020).

10 *Macdonell A. C.* Shared-Memory Optimizations for Virtual Machines : PhD dissertation / Macdonell A. Cameron. — Department of Computing Science : University of Alberta, 2011. — DOI: https://doi.org/10.7939/R3Q715.

11 Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects. Vol. 2 / D. C. Schmidt [et al.]. — John Wiley & Sons, 2013.

12 *Schmidt D. C.* Acceptor and connector // Pattern Languages of Program Design. — 1996. — Vol. 3. — P. 191–229.

13 *Schmidt D. C.*, *Cranor C. D.* Half-Sync/Half-Async // Second Pattern Languages of Programs, Monticello, Illinois. — 1995.

14 *Schmidt D. C.* Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching // Pattern Languages of Program Design. — USA : ACM Press/Addison-Wesley Publishing Co., 1995. — P. 529–545. — ISBN 0201607344.

15 The ADAPTIVE Communication Environment (ACE) [Электронный ресурс]. — URL: https://www.dre.vanderbilt.edu/~schmidt/ACE.html (дата обращения: 14.02.2020).

16 *Xiaodi K.* Interprocess Communication Mechanisms With Inter-Virtual Machine Shared Memory : Master's dissertation / Xiaodi Ke. — Department of Computing Science : University of Alberta, 2011. — DOI: https://doi.org/10.7939/R3PH6H.

17 *Zheng J.*, *Harper K. E.* Concurrency Design Patterns, Software Quality Attributes and Their Tactics // Proceedings of the 3rd International Workshop on Multicore Software Engineering. — Cape Town, South Africa : Association for Computing Machinery, 2010. — P. 40–47. — (IWMSE '10). — ISBN 9781605589640. — DOI: 10.1145/1808954.1808964. — URL: https://doi.org/10.1145/1808954.1808964.

ПРИЛОЖЕНИЕ А. ПСЕВДОКОД АЛГОРИТМОВ РАБОТЫ С МУЛЬТИПЛЕКСОРОМ СОБЫТИЙ В РАЗДЕЛЯЕМОЙ ПАМЯТИ

Листинг А.1 – Псевдокод процедуры получения оповещений из мультиплексора событий в разделяемой памяти

```
void MutliplexerServer::handle_signals() {
    // Шаг 1. Если в futex записан 0, значит, нет оповещений для
   обработки. Тогда процесс переходит в состояние сна.
    m_mux->wait();
    // Шаг 2. Атомарно получить актуальное значение futex и
   установить вместо него 0.
    int32_t futex = atomic_exchange(&m_futex, 0);
    // Шаг 3. Подсчитать количество установленных битов в числе,
   чтобы не выполнять линейное сканирование всех 32 битов.
    uint8_t cnt = popcnt(futex);
    for (uint8_t i = 0; i < cnt; i++) {
        // Шаг 4. Для каждого бита futex проверить соответствующие
   ему сигнальные числа.
        uint8_t f = get_unset_lsb(&futex);
        // Шаг 5. Атомарно получить значение сигнального числа и
   записать в него 0.
        int64_t signal = atomic_exchange(&m_signal[f], 0);
        uint8_t nsignals = popcntl(signal);
        // Шаг 6. Для каждого найденного сигнала запустить его
   обработку.
        for (uint8_t j = 0; j < nsignals; j++) {
            uint8_t s = get_unset_lsb(&signal);
            this->handle_signal(i * 64 + s);
        }
    }
}

// Выполняет обработку соединения, которому ранее был выдан номер id
void MultiplexerServer::handle_signal(Signal id);

// Возвращает количество выставленных битов в числе
uint8_t popcnt(int32_t value);
uint8_t popcntl(int64_t value);

// Сбрасывает младший бит числа и возвращает его позицию.
uint8_t get_unset_lsb(uint32_t & value);
uint8_t get_unset_lsb(uint64_t & value);
```

Листинг А.2 – Псевдокод процедуры оповещения процесса через мультиплексор событий в разделяемой памяти

```
void Multiplexer::notify(Signal id) {
    // Шаг 1. Выставить нужный бит в одном из сигнальных чисел в
   массиве. Число находится как результат деления номера соединения
   на 64, те.. число от 0 до 31, позиция бита как остаток от деления
   номера соединения на 64.
    m_signal[id / Multiplexer::c_signals_per_chunk].fetch_or(1 << id
   % Multiplexer::c_signals_per_chunk);
    // Шаг 2. Выставить бит futex, соответствующий сигнальному
   числу. Позиция нужного бита находится как результат деления
   нмоера соединения на 64, те.. число от 0 до 31.
    uint32_t futex = m_futex.fetch_or(1 << id / Multiplexer::
   c_signals_per_chunk);
    if (!futex) {
        // Шаг 3. Если предыдущее значение futex было 0, то
   попытаться разбудить один процесс, спящий на futex, системным
   вызовом futex.
        this->wakeup();
    }
}
```