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: 0

4 Полное число источников, использованных в работе: 13

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	1	2	4	4

6 Использование информационных ресурсов Internet: да, число ресурсов: 4

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 2.1.3.1, 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, 13]. 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 **(TBD: может, убрать?)** 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 **TBD: надо ли?**.

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.

Результаты работы применены в платформе для торговли на финансовых рынках Tbricks от компании Itiviti.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЦЕЛИ РАБОТЫ

1.1. Методы межпроцессного взаимодействия

1.2. Значимость методов на основе разделяемой памяти в межпроцессном взаимодействия

1.3. Необходимость в едином интерфейса доступа к методам межпроцессного взаимодействия

1.4. Существующие решения

1.5. Предыдущая работа

1.6. Критерий эффективности и постановка цели работы

Выводы по главе 1

TBD: выводы

ГЛАВА 2. РАЗРАБОТКА И РЕАЛИЗАЦИЯ МЕТОДОВ ЭФФЕКТИВНОГО ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ

2.1. Методы оповещения о появлении данных в разделяемой памяти

2.1.1. 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 обслуживания соответствующего соединения. **TBD: иллюстрацию?**

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. **TBD: иллюстрацию?**

2.1.1.1. 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.

2.1.2. 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.

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

TBD: нужна иллюстрация взаимодействия и стек модулей

а) процессу-читатель ожидает новых данных по всем своим TCP соединениям в состоянии сна в системном мультиплексоре оповещений;
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.

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

TBD: module stack, handshake TBD: нужна иллюстрация взаимодействия или какой- нибудь псевдокод

2.1.2.3. Достоинства и недостатки

Предложенный метод обладает следующими **достоинствами**:

- позволяет эффективно поддерживать множество соединений с использованием системных мультиплексоров оповещений (*select/poll/epoll*);
- позволяет процессу-читателю блокировать свое выполнение до появления оповещения;
- благодаря использованию эвристики из раздела 2.1.2.2 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 оповещения 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 оповещений, 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.

2.1.3. Мультиплексор оповещений в разделяемой памяти

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

Мультиплексор оповещений 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 оповещений, через который ему будут поступать оповещения о появлении данных в разделяемой памяти, которые он может считать и обработать.

Время жизни мультиплексора оповещений 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.

TBD: нарисовать схему с двумя процессами и файлом, отображенным в их памяти

2.1.3.1. Структура и алгоритм работы мультиплексора оповещений в разделяемой памяти

Структура мультиплексора оповещений 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 1.

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

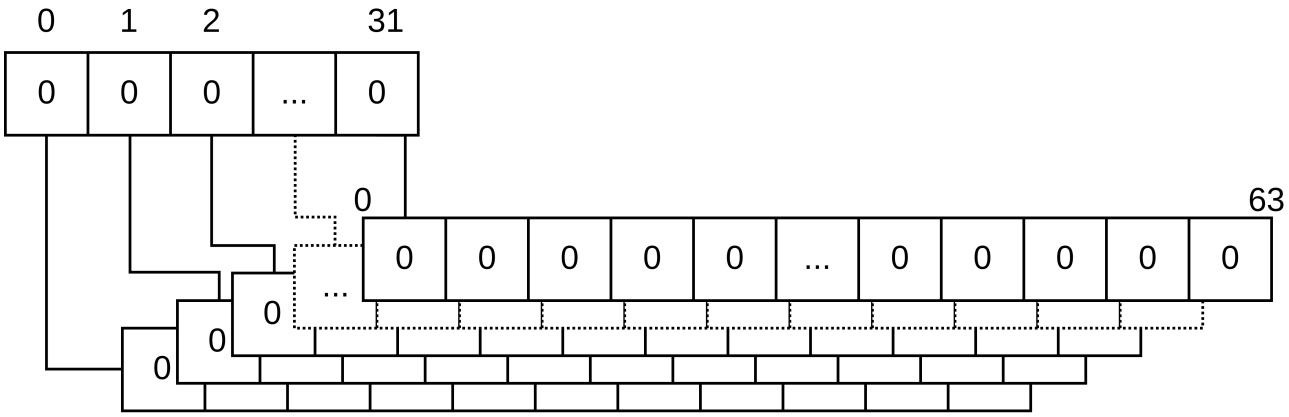

Рисунок 1 – Структура мультиплексора оповещений в разделяемой памяти

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.
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.3 и А.1.

2.1.3.2. Достоинства и недостатки

Данный метод решает проблему, описанную в разделе 2.1.1.1, позволяя определять соединение-источник сигнала за время, не зависящее от количества соединений.

Листинг 1 – 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;
};
```

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.

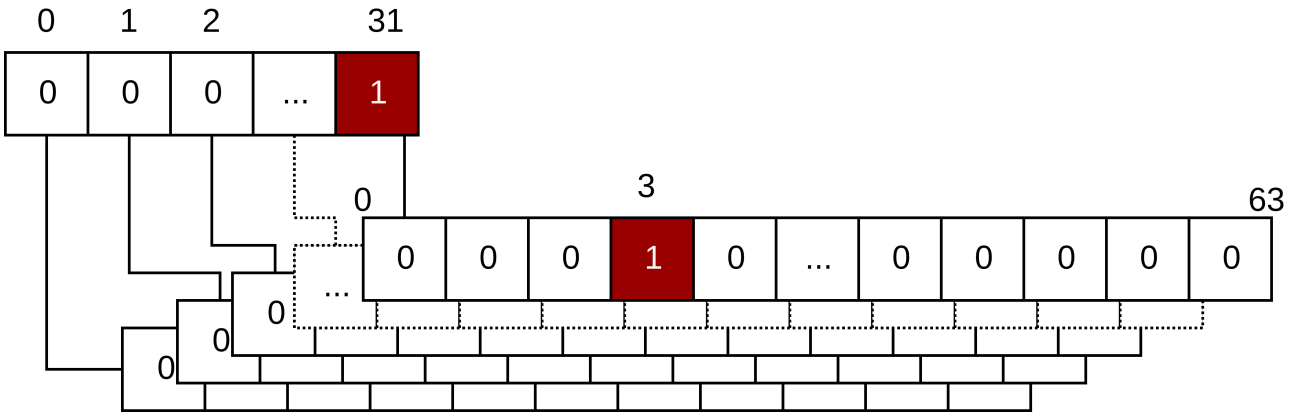

Рисунок 2 – Состояние мультиплексора оповещений в разделяемой памяти с активным сигналом #1987

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 оповещений 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.

2.1.4. Методы обслуживания соединений

Имея более совершенный механизм оповещения процессов о появлении данных в разделяемой памяти необходимо разработать также и метод обслуживания 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 обслуживание оповещений в пул потоков (полусинхронная часть) [12].

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 обслуживанию 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 обслуживанию оповещений.

2.1.4.1. Синхронный метод обслуживания соединений

В данном методе для обслуживания оповещений выделяется один поток, который обслуживает только соединения, использующие мультиплексор оповещений. Диграмма его работы представлена на рисунке 3. В отсутствие заявок поток мультиплексора находится в состоянии сна. Когда необходимо обработать соединение, ядро пробуждает этот поток и он, в свою очередь, выполняет алгоритм на листинге А.1, синхронно обрабатывая соединения, для которых были получены оповещения. Алгоритм выполняется до тех пор, пока есть оповещения для обслуживания.

Метод может быть полезен для систем с малым количеством соединений. Из-за своей простоты он может показать лучшую временную задержку на передачу данных, так как нет необходимости, как в других методах, диспетчеризовать обслуживание соединения в пуле потоков или выбирать новый поток-лидер перед обслуживанием данного соединения.

Однако, поскольку поток, обслуживающий мультиплексор, один, то когда он обслуживает текущее соединение, все остальные активные соединения простаивают. Данный метод далее в работе не рассматривается, так как не смотря на потенциально более низкую временную задержку на передачу данных при его использовании, он ограничен в количестве одновременно активных соединений.

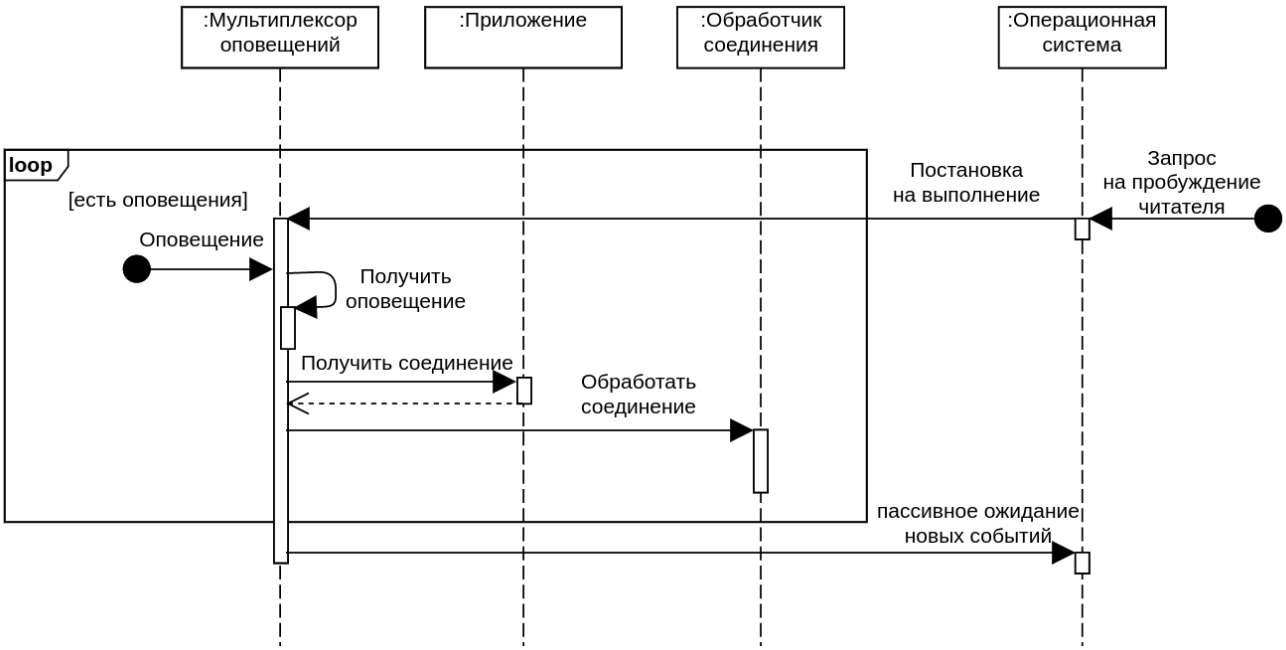

Рисунок 3 – Диаграмма синхронного обслуживания соединений выделенным потоком мультиплексора

2.1.4.2. Метод обслуживания соединений ”Полусинхронный/Полуреактивный“

Описан в ранней работе автора [1]. В данном методе оповещения обслуживает пул потоков. Пул потоков получает задачи на обслуживание от выделенного потока мультиплексора оповещений. Применение шаблона проектирования сетевых приложений ”Полусинхронный/Полуреактивный“ [12] к синхронному методу позволяет избежать простаивания активных соединений и обслуживать одновременно столько же соединений, сколько потоков выделено в пуле. Диаграмма работы метода представлена на рисунке 4.

Получение оповещений для соединения и обслуживание заявок для этого соединения происходит в разных потоках. Необходимо сохранить гарантию последовательного обслуживания заявок в соединении. То есть, в один момент времени заявки одного соединения обслуживает не более одного потока. Может произойти ситуация, при которой очередное оповещение для очередной заявки будет

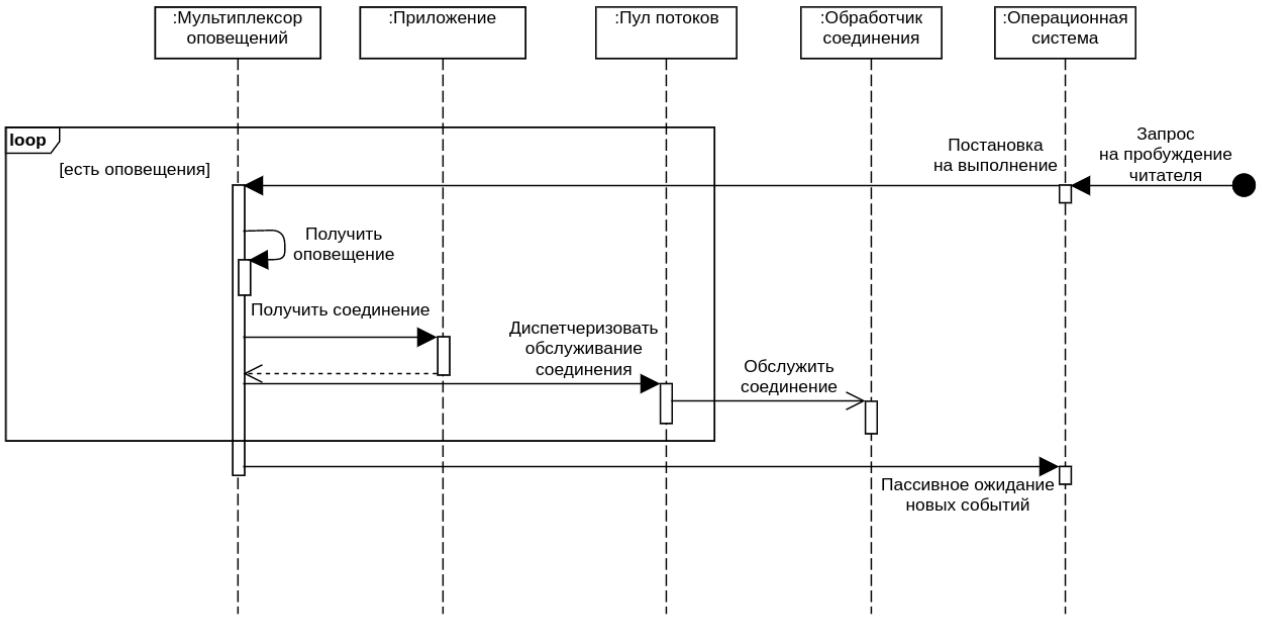

Рисунок 4 – Диаграмма диспетчеризации и обслуживания соединений по методу ”Полусинхронный/Полуреактивный“

получено во время обслуживания текущей. В таком случае, поток из пула должен обслужить новую заявку после текущей, либо обслуживание должно быть делегировано пулу потоков после обслуживания текущей заявки. Первая стратегия предпочтительнее, так как имеет лучшую локальность по памяти и времени: один и тот же поток последовательно обслуживает заявки для одного и того же соединения при их наличии. В обоих случаях необходимо синхронизировать диспетчеризацию и непосредственно процедуру обслуживания соединения.

На листингах 2 и 3 представлен исходный код процедур диспетчеризации и обслуживания соединений. Между собой процедуры синхронизированы механизмом взаимного исключения и набором состояний.

Процедура *check_close_n_handle_send()* вызывается потоком мультиплексора для диспетчеризации обслуживания соединения. В зависимости от состояния соединения, выполняется один из трех сценариев.

а) Если соединение закрыто, то игнорировать оповещение из мультиплексора. Подчисткой ресурсов соединения занимается либо поток, закрывающий соединение, либо поток, обслуживающий соединение.

б) Если соединение уже сейчас обслуживается (или же обслуживание диспетчеризовано), то необходимо через состояние соединение заставить поток из пула произвести обслуживание еще раз.

Листинг 2 – Процедура диспетчеризации обслуживания соединения

```
void IMultiplexerListener::check_close_n_handle_send()
{
    std::unique_lock<std::mutex> lock(m_input_close_mutex);
    if (m_state == State::Closed) {
        return;
    }
    if (m_state == State::Handling) {
        m_state = State::KeepHandling;
        return;
    }
    m_state = State::Handling;
    lock.unlock();

    m_thread_pool->dispatch_async([this] { this->process_send() });
}
```

Листинг 3 – Процедура обслуживания соединения в мультиплексоре оповещений

```
void IMultiplexerListener::process_send()
{
    while (true) {
        this->handle_send();

        std::unique_lock<std::mutex> lock(m_input_close_mutex);

        if (m_state == State::Closed) {
            lock.unlock();
            this->on_deferred_close();
            break;
        }
        else if (m_state == State::Handling) {
            m_state = State::Idle;
            break;
        }
        else { //m_state == State::KeepHandling
            m_state = State::Handling;
        }
    }
}
```

в) Если же соединение в данный момент не обслуживается, то пометить его таковым и диспетчеризовать обслуживание в пуле потоков.

Процедура обслуживания заявок в соединении *process_send()* выполняется потоком из пула. Перед завершением обслуживания соединения необходимо проверить, было ли получено новое оповещение (состояние соединения

KeepHandling). И если да, то повторить обслуживание. Состояние *KeepHandling* необходимо, так как иначе может произойти утеря оповещения. А именно в ситуации, когда поток мультиплексора при получении оповещения видит, что соединение еще обслуживается (состояние *Handling*), а поток из пула уже закончил фактическое обслуживание и вскоре поменяет состояние соединения на *Idle*.

Листинг 4 – Процедура закрытия соединения в мультиплексоре оповещений

```
bool IMultiplexerListener::try_deactivate_listener()
{
    std::unique_lock<std::mutex> lock(m_input_close_mutex);
    if (m_state != State::Idle) {
        lock.unlock();
        m_mux->deregister_listener(this);
        return true;
    }
    m_state = State::Closed;
    return false;
}
```

Закрытие соединения также необходимо синхронизировать с его обслуживанием. Если соединение обслуживается прямо сейчас, нельзя освобождать занятые им ресурсы. Необходимо дождаться завершения обслуживания и только потом провести подчистку. В настоящей работе предлагается в такой ситуации выполнять закрытие потоком, который в данный момент обслуживание соединение. Процедура *try_deactivate_listener()* проверяет состояние соединения: закрывать соединение и освобождать ресурсы можно только в состоянии *Idle*, в противном случае соединение помечается как закрытое, чтобы поток из пула мог завершить процедуру закрытия после завершения обслуживания соединения. Исходный код приведен на листинге 4.

Данный метод позволяет улучшить качество обслуживания соединений, используя параллелизм обслуживания на уровне соединений. Но использование дополнительных потоков при обслуживании может негативно влиять на временную задержку на передачу данных. Диспетчеризация и постановка потока из пула на выполнение для обслуживания заявки может занимать существенное время.

2.1.4.3. Метод обслуживания соединений ”Лидер/Последователи“

Как было сказано выше, в методе обслуживания соединений ”Полусинхронный/Полуреактивный“ заявки в соединении обслуживаются

потоками из пула потоков. От приема оповещения о наличии данных в разделяемой памяти до обслуживания соединения происходит диспетчеризация и постановка на выполнение потока из пула для обслуживания заявки. Использование метода ”Лидер/Последователи“ [8] позволяет избежать второго слагаемого и, возможно, получить меньшую временную задержку на передачу данных.

В рассматриваемом методе, в отличие от предыдущего ”Полусинхронный-/Полуреактивый“, отсутствует выделенный поток мультиплексора. Мультиплексированием и обслуживанием занимаются потоки из пула потоков. Это позволяет избежать накладных расходов на поддержание лишнего потока, синхронизацию между потоком мультиплексора и потоками пула, временной задержки на постановку на выполнение потока для обслуживания соединений [3, с. 41]. Диаграмма работы метода представлена на рисунке 5.

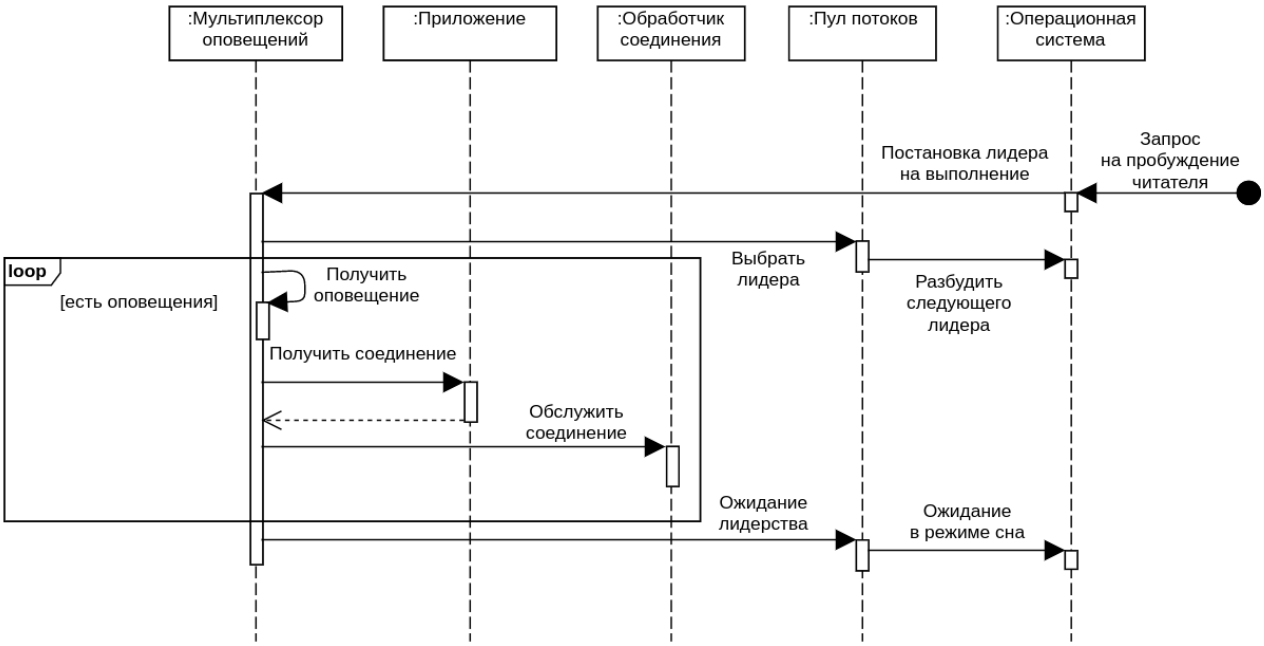

Рисунок 5 – Диаграмма обслуживания соединений по методу ”Лидер/Последователи“

В каждый момент времени состояние мультиплексора опрашивает не более одного потока, потока-лидера. При наличии оповещений для обслуживания, поток-лидер диспетчеризует в пул потоков постановку на выполнение нового лидера. Сам же он перестает быть лидером и занимается обслуживанием соединений, для которых получены оповещения. Это требует внесения изменений в процедуру опроса мультиплексора и обслуживания соединений. Изменения представлены на листинге А.2. Среди них:

а) Поток-лидер должен запомнить все соединения, которые ему необходимо обслужить.
б) Каждое из этих соединений необходимо пометить как *Handling*. Метод *should_process()* приведен на листинге 5.
в) Поток-лидер диспетчеризует нового лидера.
г) После обслуживания соединений для полученных оповещений поток переходит в состояние ожидания лидерства.

Листинг 5 – Процедура подготовления соединения к обслуживанию в модели обслуживания соединений ”Лидер/Последователи“

```
bool IMultiplexerListener::should_process()
{
    std::unique_lock<std::mutex> lock(m_input_close_mutex);
    if (m_state == State::Closed) {
        return false;
    }
    if (m_state == State::Handling) {
        m_state = State::KeepHandling;
        return false;
    }
    m_state = State::Handling;
    return true;
}
```

Второй шаг необходим, так как в противном случае новый лидер может получить новое оповещение для уже обслуживаемых предыдущим лидером соединений и тогда два потока будут параллельно обслуживать одно и то же соединение, что недопустимо.

В настоящей работе реализована вариация метода, в которой поток-лидер обслуживает соединения для всех обнаруженных оповещений. Такой подход может быть эффективен, когда число одновременно активных соединений мало. Недостаток такой реализации состоит в худшем использовании параллелизма на уровне соединений и накоплении временной задержки на передачу данных для соединений, оказавшихся в очереди на обслуживание.

2.1.4.4. Метод обслуживания соединений ”Лидер/Последователи“ с активным ожиданием

Выводы по главе 2

TBD: Вывод по главе

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 6.

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 6).

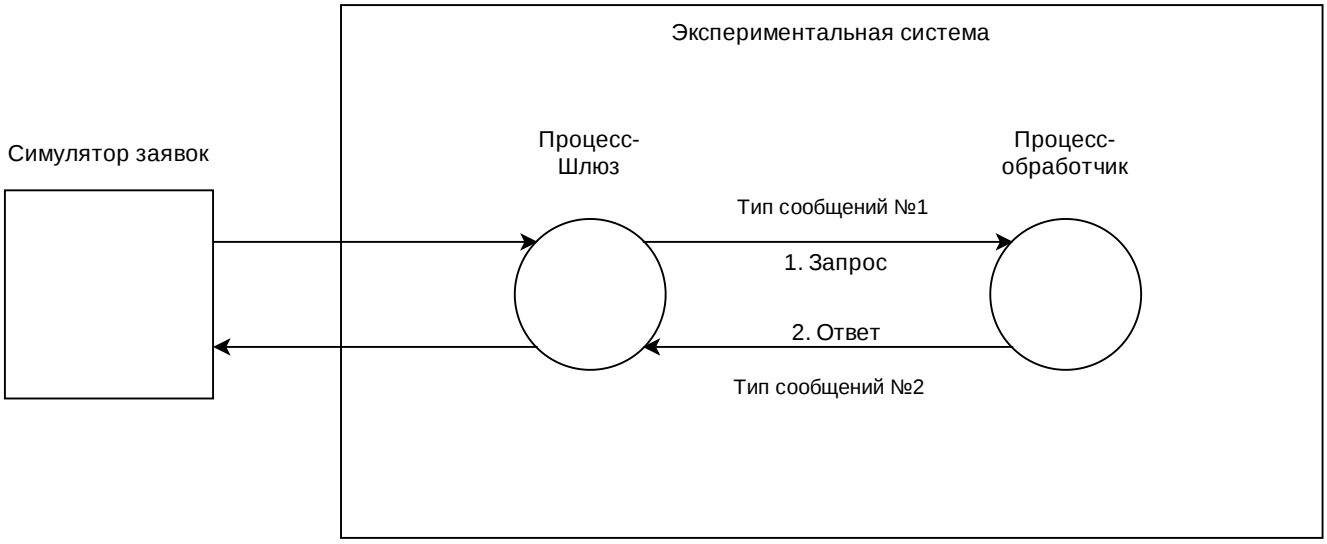

Рисунок 6 – Схема взаимодействия процессов в эксперименте

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.

TBD: надо как- то адекватно это описать Пользовательская логика процесс-обработчика в среднем отклоняет 25% заявок, в то время как 75% заявок отправляются в процесс-шлюз.

3.1.3. Используемые обозначения

— Δ – временная задержка между сериями заявок;
— δ – временная задержка между заявками в серии;
— τ – временная задержка на передачу данных;
— T – время обслуживания заявки.

— транспортный поток – поток из пула транспортных потоков, непосредственно обслуживающий получаемые заявки.

Из-за сложного характера распределений для представления полученных данных используются гистограммы выборок и 50, 80, 90, 95 и 99 процентили.

3.1.4. Характер экспериментальной нагрузки

Характеристики потока заявок, создаваемого симулятором, приведены в таблице 1.

Таблица 1 – Процентили интервалов между сериями заявок и заявками, создаваемыми симулятором

Процентиль	0%	50%	80%	90%	95%	99%	100%
Δ, мс	5	9	11	12.5	13	15	109
δ, мкс	52	62	82	110	115	128	4819

3.1.5. Время обслуживания заявок в процессах

На рисунке 7 представлена гистограмма времени обслуживания заявок в процессе-обработчике. В таблице 2 представлены процентили времени обслуживания заявок в процессе-обработчике.

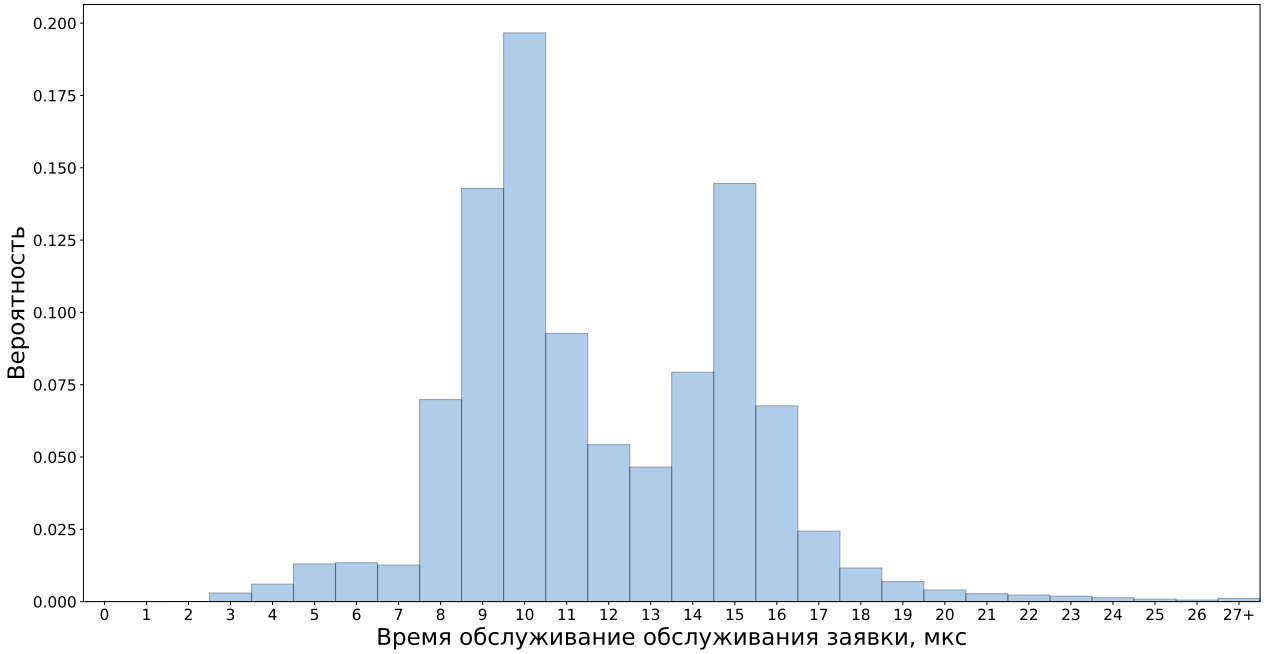

Рисунок 7 – Гистограмма времени обслуживания заявки в процессе-обработчике

Таблица 2 – Процентили времени обслуживания заявок в процессе-обработчике

Процентиль	0%	50%	80%	90%	95%	99%	100%
T, мкс	2	11	15	16	17	21	285

На рисунке 8 представлена гистограмма времени обслуживания заявок в процессе-шлюзе. В таблице 3 представлены процентили времени обслуживания заявок в процессе-обработчике.

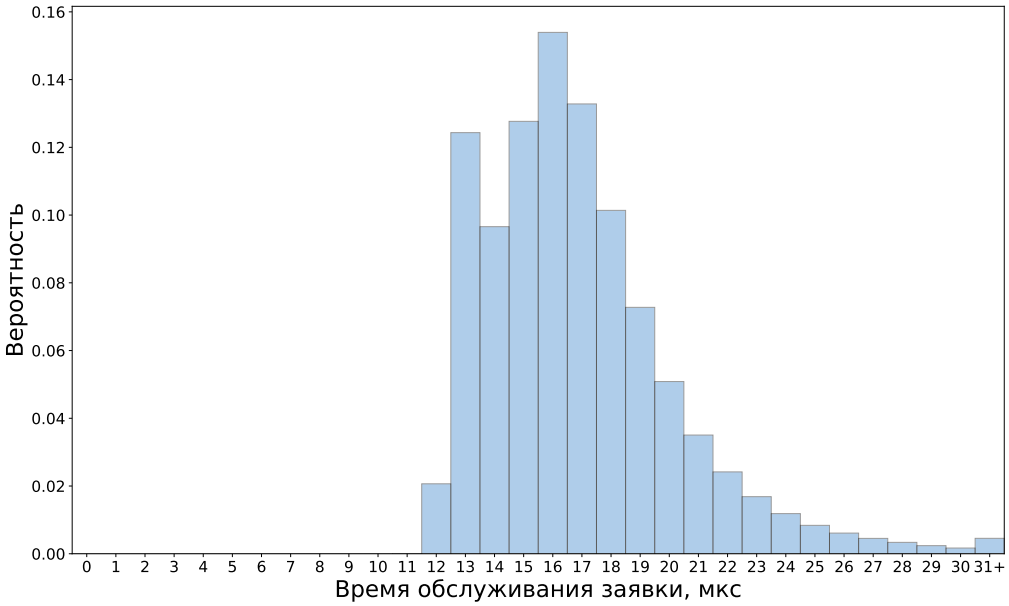

Рисунок 8 – Гистограмма времени обслуживания заявки в процессе-шлюзе

Таблица 3 – Процентили времени обслуживания заявок в процессе-шлюзе

Процентиль	0%	50%	80%	90%	95%	99%	100%
T, мкс	11	16	19	21	23	27	5678

TBD: а надо ли мне вообще говорить тогда про время обслуживания заявок в процессе-шлюзе? Может, убрать совсем? 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 на обслуживание влияет на данный показатель.

3.2. Использование TCP для передачи данных

В качестве точки отсчета в настоящей работе выступает метод межпроцессного взаимодействия на основе TCP, используемый посредством сокетов **TBD: ссылка на определение ??**.

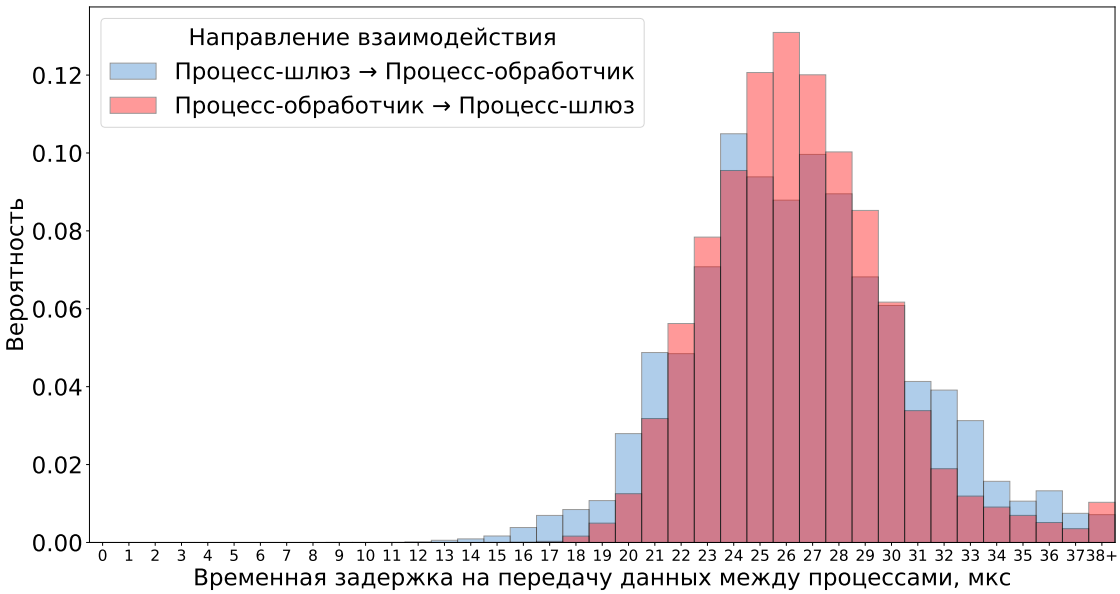

Рисунок 9 – Гистограмма временной задержки на передачу данных между процессами при использовании TCP

Гистограмма временной задержки на передачу данных для данного метода приведена на рисунке 9. В таблице 4 приведены основные временные характеристики данного метода.

Таблица 4 – Основные показатели временной задержки на передачу данных для метода на основе TCP

Направление взаимодействия/ Показатель	Процесс-шлюз → Процесс-обработчик	Процесс-обработчик → Процесс-шлюз
$\min(\tau)$, мкс	9	13
50%, мкс	26	26
80%, мкс	30	29
90%, мкс	32	30
95%, мкс	34	32
99%, мкс	37	38
$\max(\tau)$, мс	2.1	9.2

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

В данном подразделе приведены данные об экспериментах с методом межпроцессного взаимодействия, описанным в разделе 2.1.2.

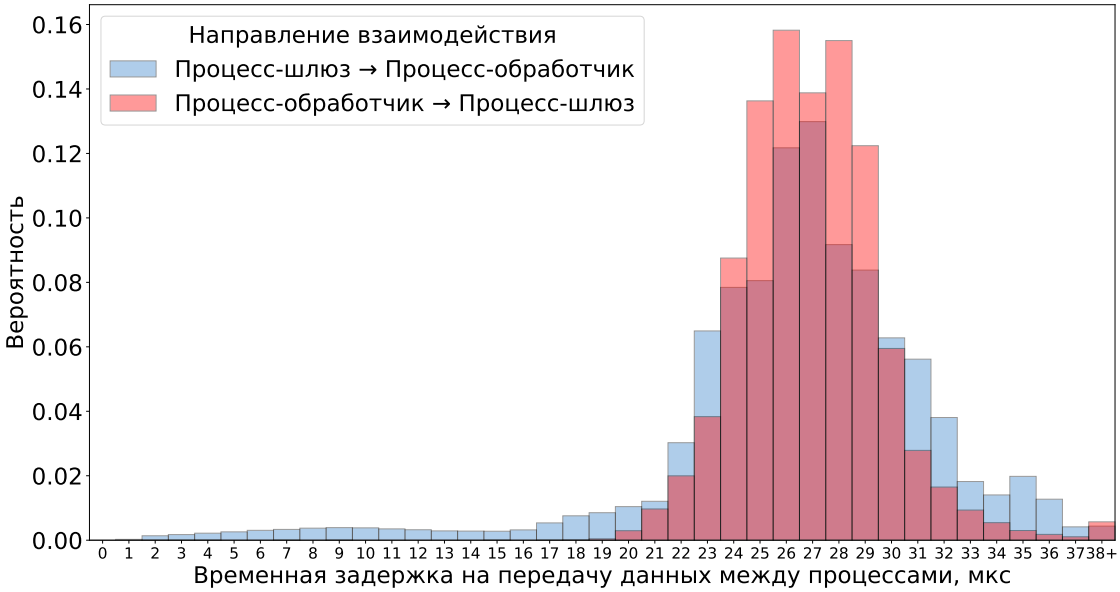

Рисунок 10 – Гистограмма временной задержки на передачу данных между процессами при использовании разделяемой памяти для передачи данных и TCP для оповещения о появлении данных в ней

В таблице 5 приведены основные временные характеристики данного метода. На рисунке 10 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 2.1.2.2. А именно принять и начать обслуживание очеред-

Таблица 5 – 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	2
50%, мкс	27	27
80%, мкс	30	29
90%, мкс	32	30
95%, мкс	34	31
99%, мкс	36	35
max(t), мс	3	9.8

ной заявки, не используя дорогостоящий механизм оповещения, если заявка уже находится в очереди в разделяемой памяти.

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 2.1.3.

3.3.2.1. Методы с пассивным ожиданием оповещений

В методах с пассивным ожиданием оповещений поток мультиплексора событий использует примитив *futex* **TBD:Может, сослаться на определение futex?** для ожидания новых сигналов (см. разделы 2.1.4.2 и 2.1.4.3). 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” В данном подразделе приведены данные об экспериментах с методом межпроцессного взаимодействия, описанным в разделе 2.1.4.2.

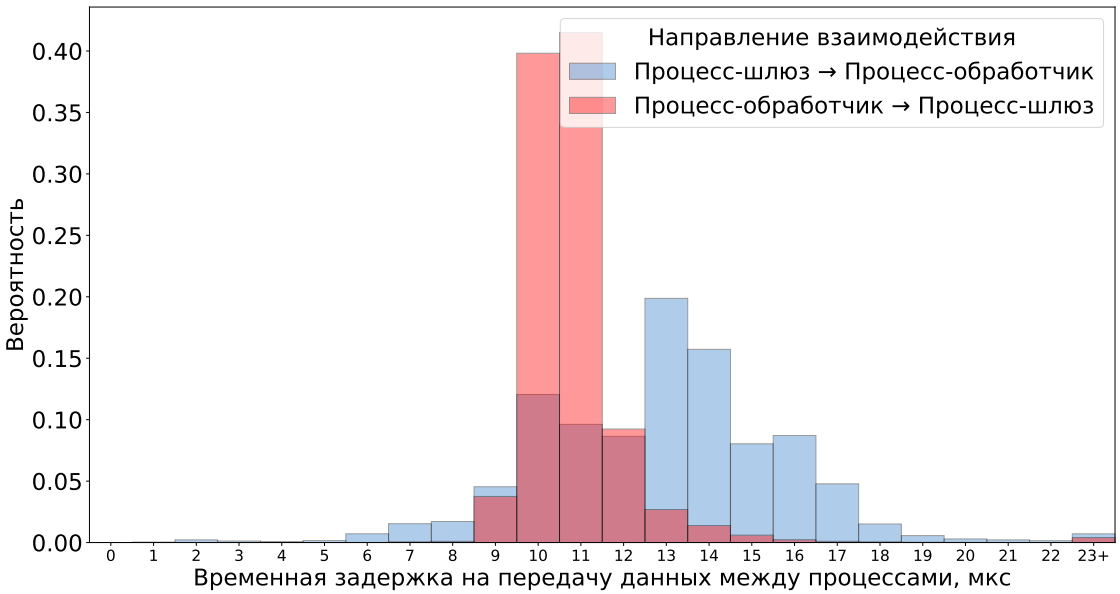

Рисунок 11 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, пассивное ожидание событий на мультиплексоре событий в разделяемой памяти и метод ”Полусинхронный/Полуреактивный“ при обслуживании заявок

В таблице 6 приведены основные временные характеристики данного метода. На рисунке 11 приведена гистограмма временной задержки на передачу данных для данного метода.

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 7), а потому в среднем временная задержка на передачу данных имеет худшие показатели, так как это может задерживать прием очередных заявок.

Таблица 6 – 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
50%, мкс	13	11
80%, мкс	15	11
90%, мкс	16	12
95%, мкс	17	13
99%, мкс	21	15
max(t), мс	6.9	11.6

Диспетчеризация и обработка соединений по модели ”Лидер/Последователи”

В данном подразделе приведены данные об экспериментах с методом межпроцессного взаимодействия, описанным в разделе 2.1.4.3.

В таблице 7 приведены основные временные характеристики данного метода.

На рисунке 12 приведена гистограмма временной задержки на передачу данных для данного метода.

Таблица 7 – 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
50%, мкс	10	9
80%, мкс	13	9
90%, мкс	15	10
95%, мкс	17	10
99%, мкс	19	12
max(t), мс	2.4	9.5

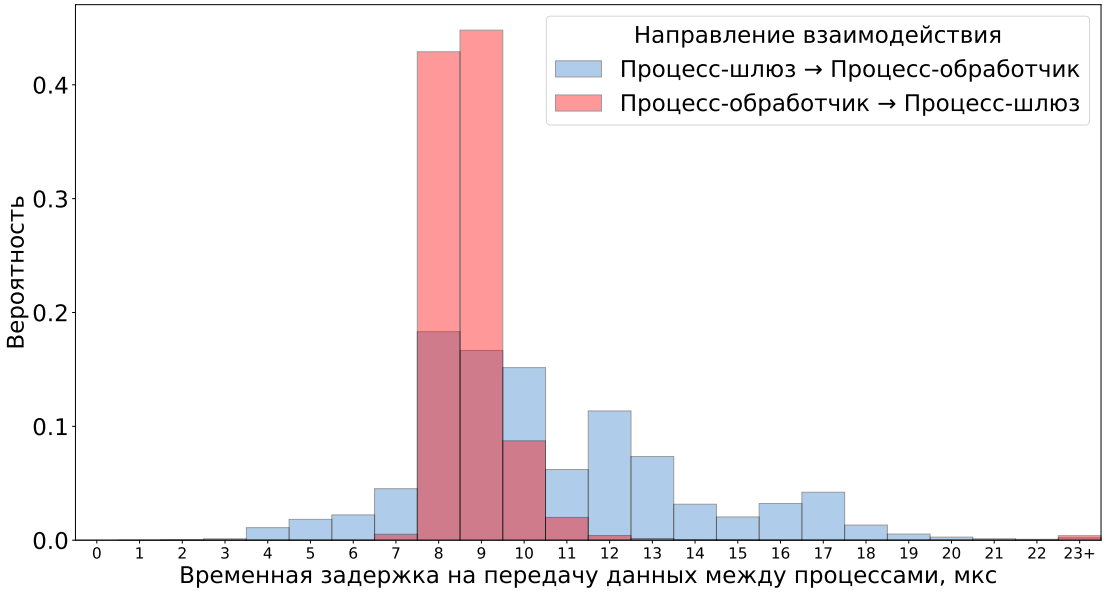

Рисунок 12 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, пассивное ожидание оповещений в мультиплексоре событий в разделяемой памяти и метод ”Лидер/Последователи“ при обслуживании заявок

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 2.1.4.4). 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”
В данном подразделе приведены данные об экспериментах с методом межпроцессного взаимодействия, описанным в разделе 2.1.4.4.

В таблице 8 приведены основные временные характеристики данного метода. На рисунке 13 приведена гистограмма временной задержки на передачу данных для данного метода.

Таблица 8 – 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
50%, мкс	4	4
80%, мкс	5	5
90%, мкс	6	5
95%, мкс	7	6
99%, мкс	10	6
max(t), мс	0.064	0.017

Выводы по исследованию метода с активным ожиданием Метод с активным ожиданием событий в мультиплексоре в разделяемой памяти ожидаемо показы-

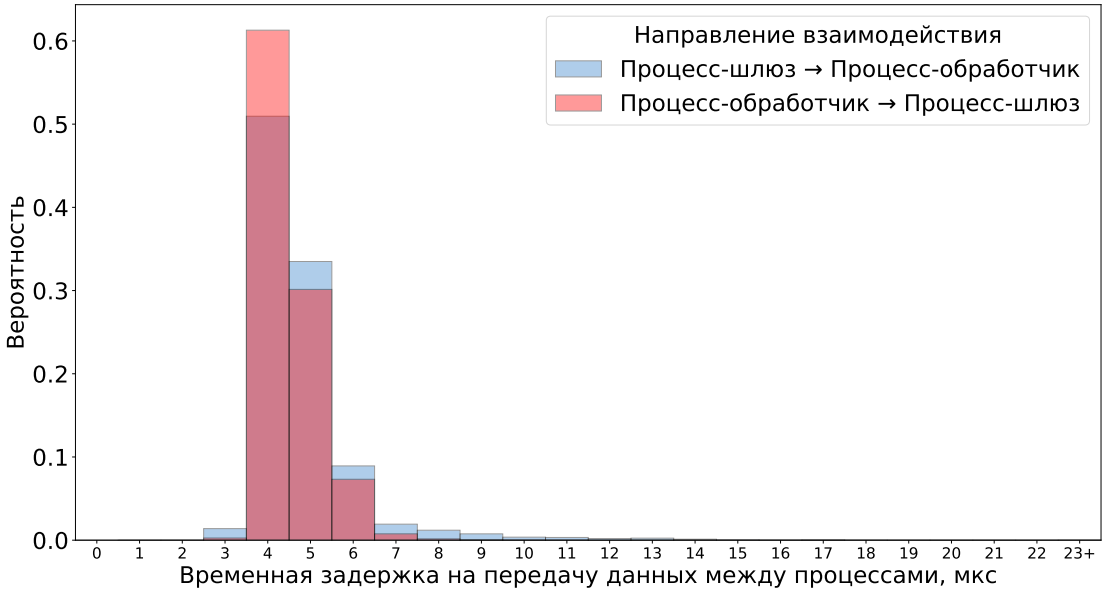

Рисунок 13 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, активно опрашиваемый мультиплексор в разделяемой памяти и модель ”Лидер/Последователи“ при обслуживании заявок

вает 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 пассивным ожиданием, пробуждение потока до постановки его на выполнение занимает около 5-6 мкс. **TBD: а надо ли оно мне? Вопрос корректности приведения циклов на AMD к секундам спорный** 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, *17 мкс* и *10 мкс* для 95 процентиля для пассивного варианта ”Лидер/Последователи“ против *34 мкс* и *31 мкс* для 95 процентиля 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 *17 мкс* и *10 мкс* для 95 процентиля против *17 мкс* и *13 мкс* для 95 процентиля при использовании метода обслуживания заявок ”Полусинхронный/Полуреактивный“.

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, *7 мкс* и *6 мкс* для 95 процентиля.

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, с интервалом примерно 10 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 *Schmidt D. C.* Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects. — Wiley, 2000. — С. 633. — ISBN 0471606952.

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

13 *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.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД АЛГОРИТМОВ РАБОТЫ С МУЛЬТИПЛЕКСОРОМ СОБЫТИЙ В РАЗДЕЛЯЕМОЙ ПАМЯТИ

Листинг А.1 – Исходный код 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 позицию этого бита.
uint8_t get_unset_lsb(uint32_t & value);
uint8_t get_unset_lsb(uint64_t & value);
```

Листинг А.2 – Исходный код процедуры получения оповещений из мультиплексора событий в разделяемой памяти для метода обслуживания соединений ”Лидер/Последователи“

```
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();
    int32_t futex = atomic_exchange(&m_futex, 0);
    std::vector<int32_t> signals;
    listeners.reserve(Multiplexer::c_signals_per_mux);
    uint8_t cnt = popcnt(futex);
    for (uint8_t i = 0; i < cnt; i++) {
        uint8_t f = get_unset_lsb(&futex);
        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);
            // Для каждого полученного оповещения
            // отметить соединение как Handling,
            // либо проигнорировать оповещение
            if (this->should_handle(i * 64 + s)) {
                signals.emplace_back(i * 64 + s);
            }
        }
    }

    // Создать нового лидера, который будет
    // выполнять процедуру handle_signals следующим
    m_thread_pool->promote_new_leader();

    // Непосредственно обслуживание соединений по полученным
   оповещениям
    for (int32_t signal : signals) {
        this->handle_signal(i * 64 + s);
    }
}

// Возвращает true, если соединение было Idle, и помечает его
   Handling
// Если соединение в состоянии Handling, то переводит его в
   состояние KeepHandling
// Возвращает false, если любое другое состояние.
bool MultiplexerServer::should_handle(Signal id);
```

Листинг А.3 – Исходный код 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();
    }
}
```