

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ»

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. _____

К защите допустить

Руководитель ОП Бессмертный И.А., доцент, д.т.н. _____

« ____ » _____ 20__ г.

Санкт-Петербург, 2020 г.

Студент Губарев В.Ю.

Группа Р42111 Факультет ПИиКТ

Направленность (профиль), специализация
”Информационно-вычислительные 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
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ»

УТВЕРЖДАЮ

Руководитель ОП

доцент, д.т.н. Бессмертный И.А. _____

« ____ » _____ 20 ____ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студент Губарев В.Ю.

Группа Р42111 Факультет ПИиКТ

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

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

Направление подготовки (специальность): 09.04.04 "Программная инженерия"

Направленность (профиль): "Информационно-вычислительные системы"

Квалификация: Магистр

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 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
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,
МЕХАНИКИ И ОПТИКИ»

АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студент: 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: 16

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	4	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++	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. — URL: <https://kmu.itmo.ru/digests/article/3100> (дата обр. 02.05.2020).

Студент Губарев В.Ю. _____

Руководитель Косяков М.С. _____

« ____ » _____ 20__ г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	7
ГЛАВА 1. Обзор предметной области и постановка цели работы.....	10
1.1. Методы межпроцессного взаимодействия.....	10
1.2. Существующие решения	10
1.3. Предыдущая работа.....	10
1.4. Значимость методов на основе разделяемой памяти в межпроцессном взаимодействии	10
1.5. Необходимость в едином интерфейса доступа к методам межпроцессного взаимодействия.....	10
1.6. Формулировка проблемы.....	10
1.7. Критерий эффективности и постановка цели работы.....	10
Выводы по главе 1	10
ГЛАВА 2. Разработка и реализация методов эффективного взаимодействия процессов	11
2.1. Интерфейс доступа к разрабатываемым методам межпроцессного взаимодействия.....	11
2.2. Базовый метод межпроцессного взаимодействия на основе ТСР	12
2.2.1. Мультиплексирование соединений	13
2.2.2. Обслуживание активных соединений.....	13
2.2.3. Динамическое конфигурирование соединений	13
2.3. Применение разделяемой памяти для передачи данных	13
2.4. Методы оповещения о появлении данных в разделяемой памяти	14
2.4.1. Наивные алгоритмы в разделяемой памяти.....	14
2.4.2. ТСР	15
2.4.3. Мультиплексор событий в разделяемой памяти	17
2.4.4. Методы обработки оповещений в мультиплексоре событий.....	20
Выводы по главе 2.....	21

ГЛАВА 3. Экспериментальное исследование и обработка результатов.....	22
3.1. Постановка эксперимента.....	22
3.1.1. Конфигурация экспериментального стенда.....	22
3.1.2. Конфигурация экспериментальной системы.....	22
3.1.3. Используемые обозначения	23
3.1.4. Характер экспериментальной нагрузки.....	24
3.1.5. Время обслуживания заявок в процессах.....	24
3.2. Использование ТСП для передачи данных.....	25
3.3. Использование разделяемой памяти для передачи данных ...	26
3.3.1. Использование ТСП для оповещения о появлении данных.....	26
3.3.2. Использование мультиплексора в разделяемой памяти для оповещения о появлении данных	28
Выводы по главе 3.....	33
ЗАКЛЮЧЕНИЕ	34
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	35
ПРИЛОЖЕНИЕ А. Псевдокод алгоритмов работы с мультиплексором событий в разделяемой памяти.....	37

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, 15]. Для связи программных модулей, исполняющихся в разных процессах для обеспечения отказоустойчи-

ности 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, снижает 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, экспериментальную оценку разработанных методов эффективного 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

TBD.

- а) shmem + mux + futex + lf;
- б) shmem + mux + futex + hsha;
- в) shmem + mux + busy wait.

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

1.1. Методы межпроцессного взаимодействия

1.2. Существующие решения

1.3. Предыдущая работа

1.4. Значимость методов на основе разделяемой памяти в межпроцессном взаимодействии

1.5. Необходимость в едином интерфейсе доступа к методам межпроцессного взаимодействия

1.6. Формулировка проблемы

1.7. Критерий эффективности и постановка цели работы

Выводы по главе 1

TBD: выводы

ГЛАВА 2. РАЗРАБОТКА И РЕАЛИЗАЦИЯ МЕТОДОВ ЭФФЕКТИВНОГО ВЗАИМОДЕЙСТВИЯ ПРОЦЕССОВ

2.1. Интерфейс доступа к разрабатываемым методам межпроцессного взаимодействия

TBD: можно взять из бакалаврской ВКР или оставить на нее ссылку.

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

TBD: определение или ссылку, 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 Ассертор и Коннектор [11], которые при необходимости создают экземпляры пользовательских обработчиков соединений, реализующих интерфейс, представленный на Листинге 1.

TBD: кривовато звучит 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, исполь-

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;
};
```

зование этого шаблона позволяет разделить пользовательскую логику и нижележащие методы межпроцессного 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 закрывает все TCP-соединения процесса и другие стороны межпроцессного взаимодействия смогут корректно обработать закрытие соединений.

2.2.1. Мультиплексирование соединений

Межпроцессное взаимодействие посредством ТСП позволяет использовать системные мультиплексоры событий *select/poll/epoll* для отслеживания событий в множестве ТСП-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 [13, 16] для диспетчеризации множества соединений. В настоящей работе используется реализаций шаблона Реактор из библиотеке ACE [14], 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;
};
```

TBD

2.2.2. Обслуживание активных соединений

TBD

2.2.3. Динамическое конфигурирование соединений

TBD: надо написать про стек модулей

2.3. Применение разделяемой памяти для передачи данных

TBD: описать SPSC очередь на буфере постоянного размера

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 находиться множество взаимодействующих процессов с большим количеством соединений между ними. Соответственно, для каждого соединения 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. **TBD: иллюстрацию?**

Алгоритм №2 При большем количестве соединений возможно использовать группу выделенных потоков, активно опрашивающих некоторое количество очередей в разделяемой памяти. Например, 1 поток, постоянно опрашивающий до 10 соединений. **TBD: иллюстрацию?**

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. Кро-

ме того, постоянно работающий поток может быть вытеснен с процессора планировщиком операционной системы из-за израсходования отведенного ему кванта процессорного времени.

2.4.2. TSP

Как было сказано выше, TSP 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 TSP для оповещения о появлении данных

TBD: нужна иллюстрация взаимодействия и стек модулей

- а) процессу-читатель ожидает новых данных по всем своим TSP 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 демultipлексирует активное TSP 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 TSP.

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

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

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 ТСП 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 ТСП, в том, как передается само сообщения. Но кроме использования среды для передачи самих данных выполняется ряд системных вызовов для оповещения процесса-читателя, каждый из которых может вносить существенную временную задержку:

- *write* – для записи 1 байта в ТСП-сокеты процессом-писателем;
- *select/poll/epoll* – для демultipлексирования нужного события среди множества источников процессом-читателем.
- *read* – для чтения 1 байта из ТСП-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.

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.

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

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-читателя о наличии данных в очереди в разделяемой памяти, ему необходимо:

- а) атомарно выставить бит в сигнальном числе, соответствующий этому соединению;

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 состоятельности между атомарными
    операциями над массивом сигнальных чисел и над futex, массив
    выровнен на размер кэш-линии- процессора -- 64 байта.
    alignas(64) std::array<std::atomic<Signal>, c_num_chunks>
    m_signals;
};

```

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 процессавписателя – на пробуждение потока, так и со стороны процессачитателя – 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.

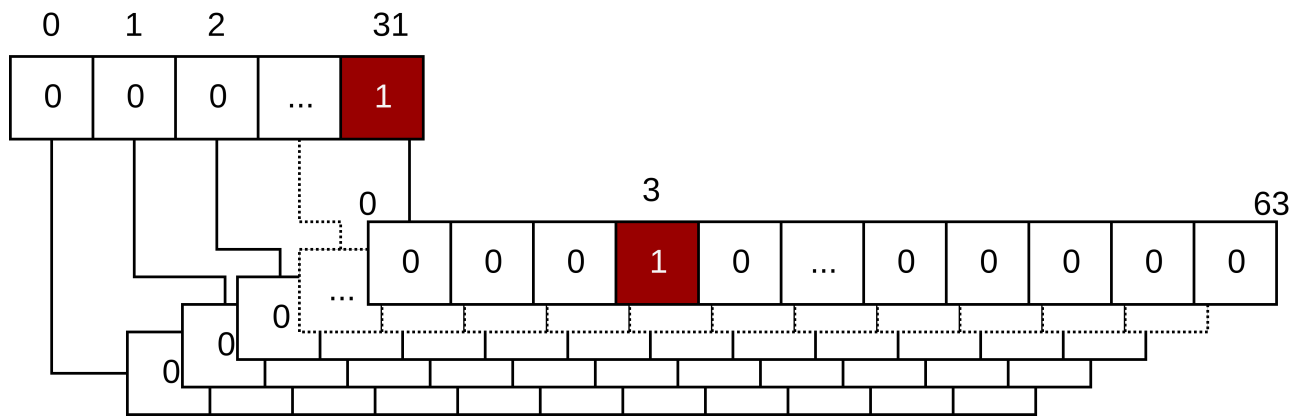


Рисунок 2 – Состояние мультиплексора событий в разделяемой памяти с активным сигналом #1987

Недостатком данного метода является механизм управления 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) [12].
- в) "Лидер/Последователи" – в один момент времени только один поток-лидер отслеживает состояние мультиплексора в блокирующем 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 [8].

2.4.4.1. Синхронный метод обработки соединений

2.4.4.2. Метод обработки соединений "Полусинхронный/Полуреактивный"

Описан в ранней работе [1] автора.

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

TBD: Вывод по очередной 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 3.

В настоящей работе измеряется 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).

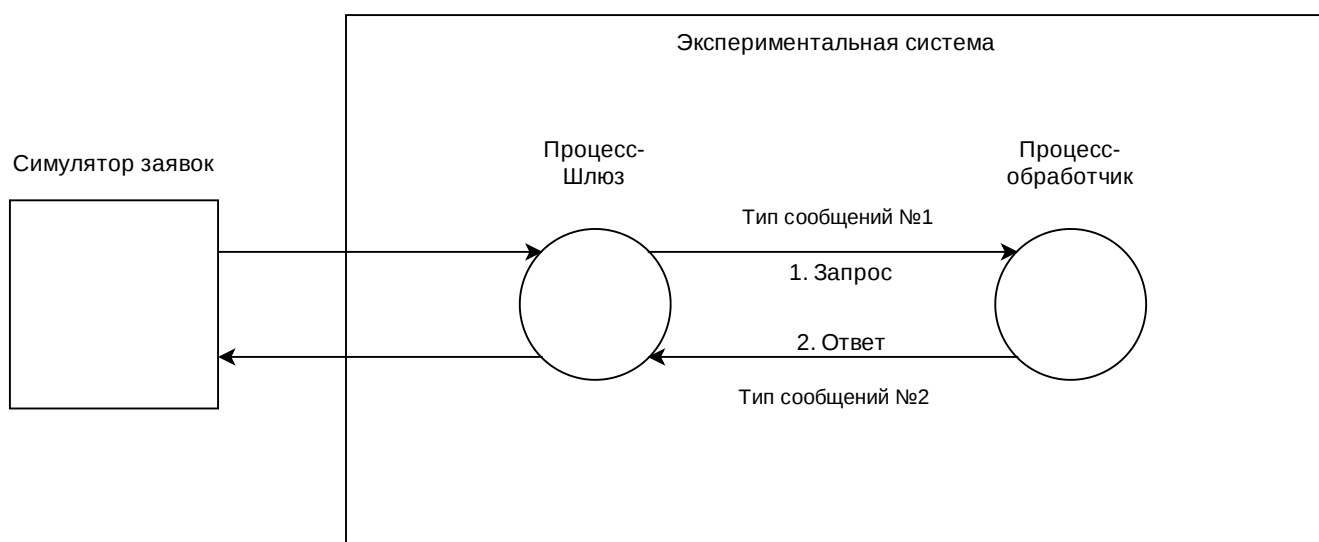


Рисунок 3 – Схема взаимодействия процессов в эксперименте

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 **TBD: Дать определение транспортному потоку**. 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

TBD: а надо ли оно мне?

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

— Δ – временная задержка между сериями заявок

— δ – временная задержка между заявками в серии

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

TBD: разобраться с обработкой соединений и обслуживанием заявок. Должно быть одно слово для обозначения одной сущности.

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

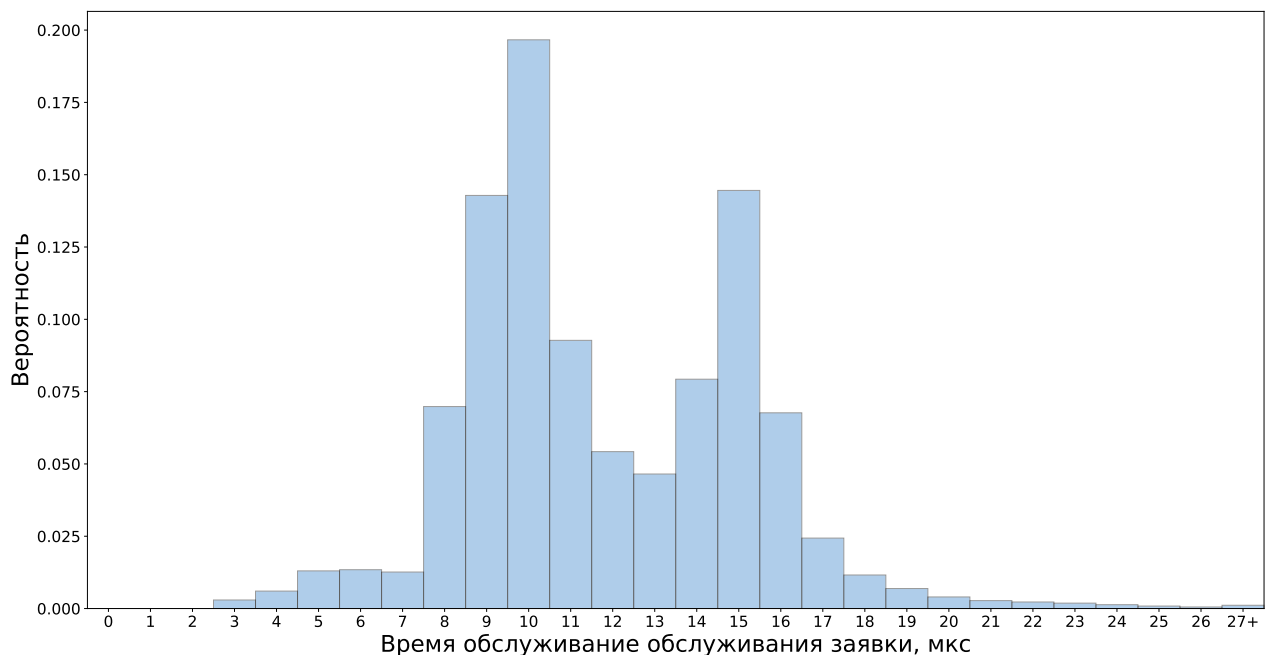


Рисунок 4 – Гистограмма времени обслуживания заявки в процессе-обработчике

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

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

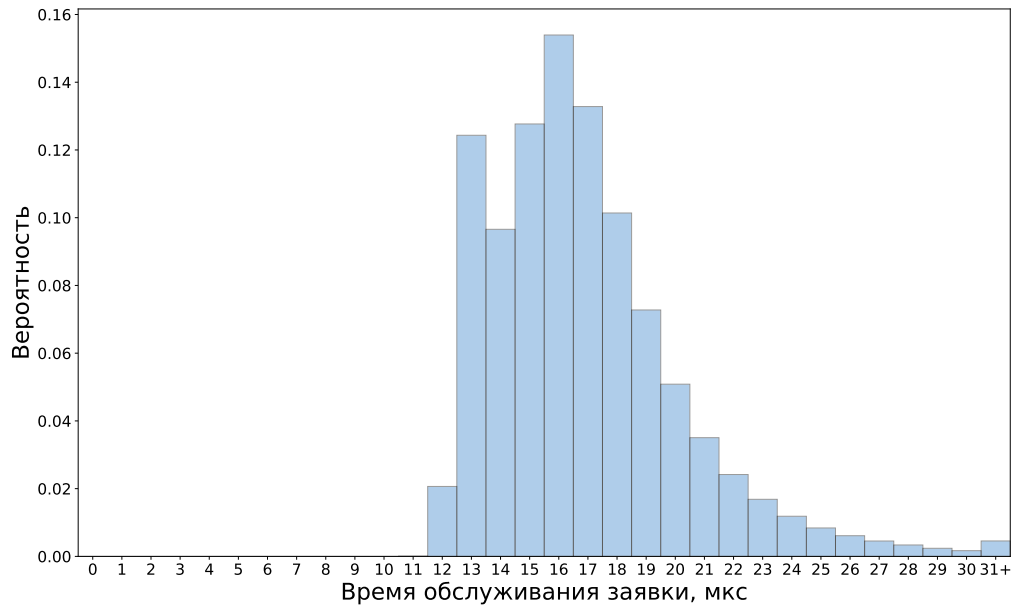


Рисунок 5 – Гистограмма времени обслуживания заявки в процессе-шлюзе

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.

3.2. Использование ТСП для передачи данных

В качестве точки отсчета в настоящей работе выступает метод межпроцессного взаимодействия на основе ТСП, используемый посредством сокетов **TBD: ссылка на определение и на раздел диссертации 2.2.**

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

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

Временная задержка на передачу данных в обоих направлениях имеет схожие значения. Это объясняется тем, что накладные расходы на использование ТСП через механизм сокетов имеют подавляющее значение над прочими факторами, влияющими на межпроцессное взаимодействие.

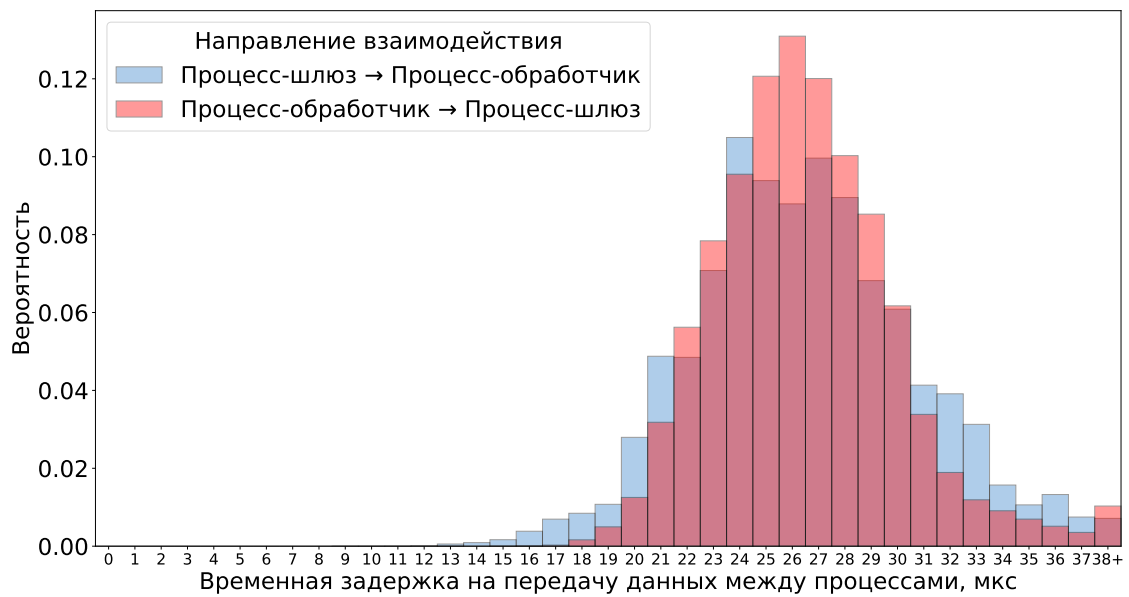


Рисунок 6 – Гистограмма временной задержки на передачу данных между процессами при использовании ТСР

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

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

3.3. Использование разделяемой памяти для передачи данных

3.3.1. Использование ТСР 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 меж-процессного 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.

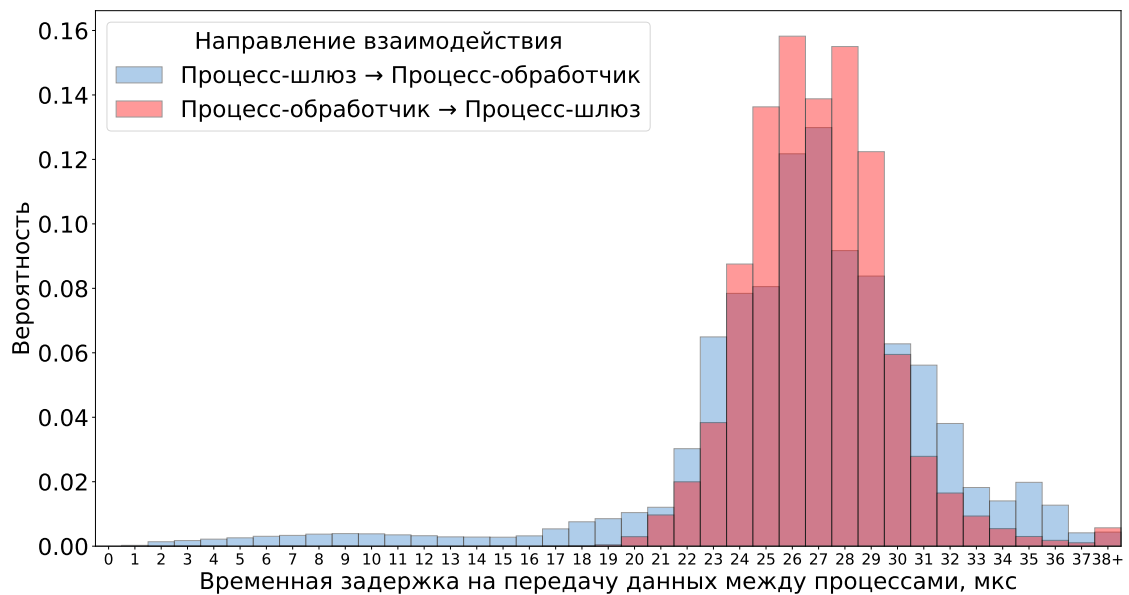


Рисунок 7 – Гистограмма временной задержки на передачу данных между процессами при использовании разделяемой памяти для передачи данных и ТСП для оповещения о появлении данных в ней

Таблица 2 – Основные показатели временной задержки на передачу данных для метода, использующего разделяемую памяти для передачи данных и ТСП 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

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.

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 ТСР, 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. Блокирующие методы

В блокирующих методах поток мультиплексора событий использует примитив *futex* **TBD:Может, сослаться на определение futex?** для пассивного ожидания новых сигналов (см. Разделы 2.4.4.2 и 2.4.4.3).

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, блокирующий мультиплексор 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 ± 5.5	11.5 ± 2.5
$\max(t)$, мс	6.9	11.6
Δ , мс	10 ± 4	10 ± 4
δ , мкс	51 ± 28	91 ± 36

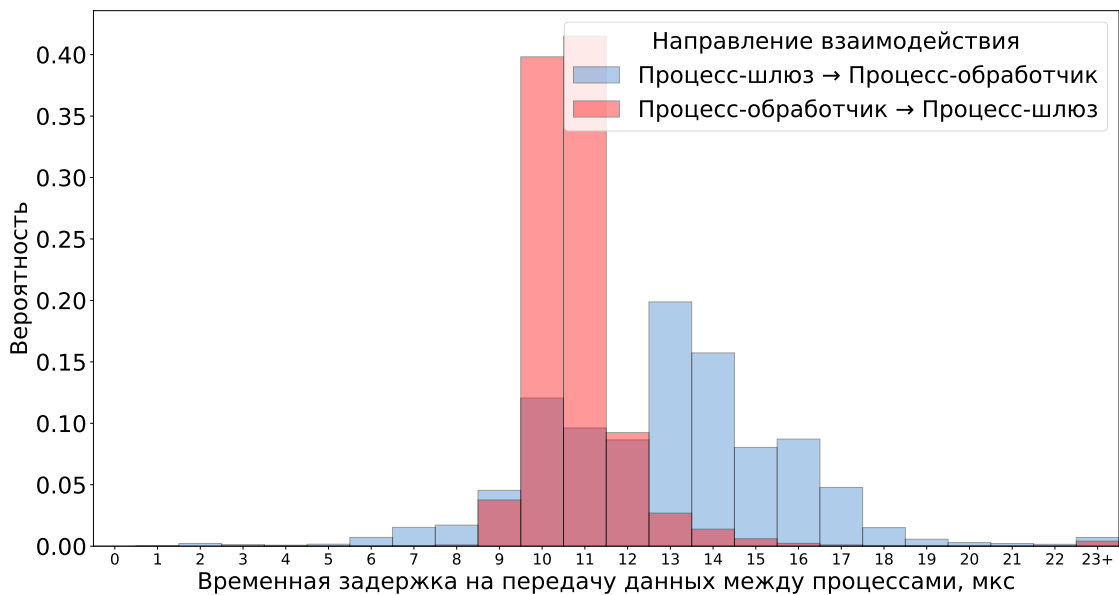


Рисунок 8 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, блокирующий мультиплексор в разделяемой памяти и метод ”Полусинхронный/Полуреактивный“ при обслуживании заявок

В Таблице 3 приведены основные временные характеристики данного метода. На Рисунке 8 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 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: $\Delta \in 8.4 \pm 5.3$ мс, $\delta \in 52 \pm 28$ мкс.

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

В Таблице 4 приведены основные временные характеристики данного метода. **TBD**

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

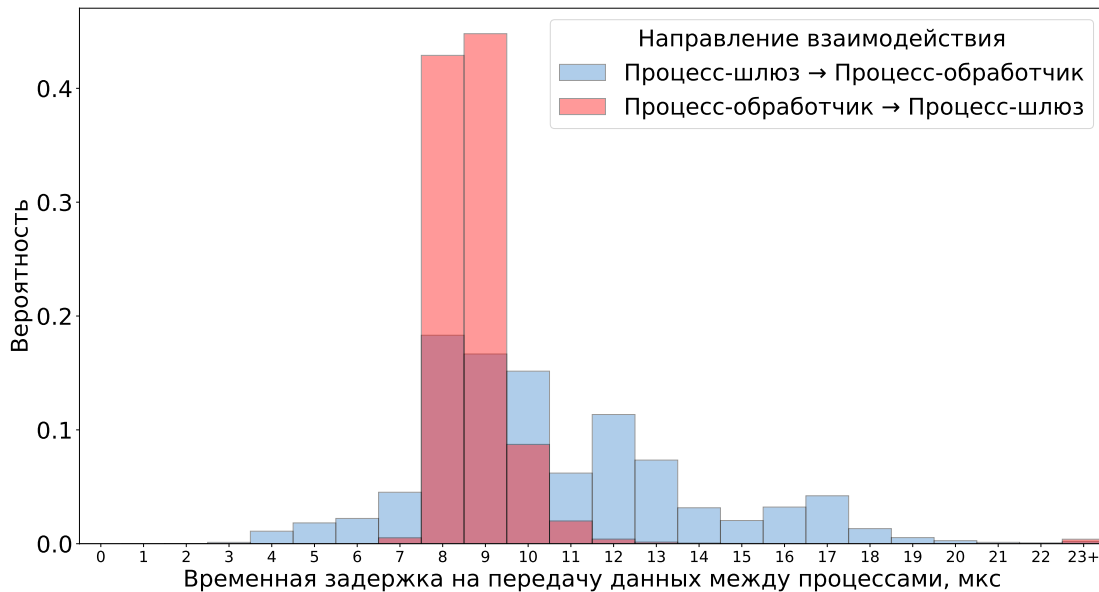


Рисунок 9 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, блокирующий мультиплексор в разделяемой памяти и метод ”Лидер/Последователи“ при обслуживании заявок

Таблица 4 – Основные показатели временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, блокирующий мультиплексор 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 ± 6.5	9.5 ± 1.5
$\max(t)$, мс	2.4	9.5
Δ , мс	8.4 ± 5.3	8.4 ± 5.3
δ , мкс	52 ± 29	90 ± 35

Аналогичный эффект как при использовании предыдущего метода наблюдается и при использовании метода "Лидер/Последователи". Отличие состоит в меньшей временной задержке на передачу данных в пределах 1-2 мкс.

3.3.2.2. Неблокирующий метод

В неблокирующем методе поток мультиплексора событий метод активного ожидания новых сигналов (см. Раздел 2.4.4.4). В данном параграфе рассматривается исключительно метод обслуживания заявок "Лидер/Последователи" т.к. в параграфе выше она показала 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: $\Delta \in 7.1 \pm 5.8$ мс, $\delta \in 83 \pm 61$ мкс.

В данном подразделе приведены данные об экспериментах с методом меж-процессного 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.

Таблица 5 – Основные показатели временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, неблокирующий 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	7.1 ± 5.8	7.1 ± 5.8
δ , мкс	220 ± 197	326 ± 272

Метод с активным ожиданием событий на 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 заяв-

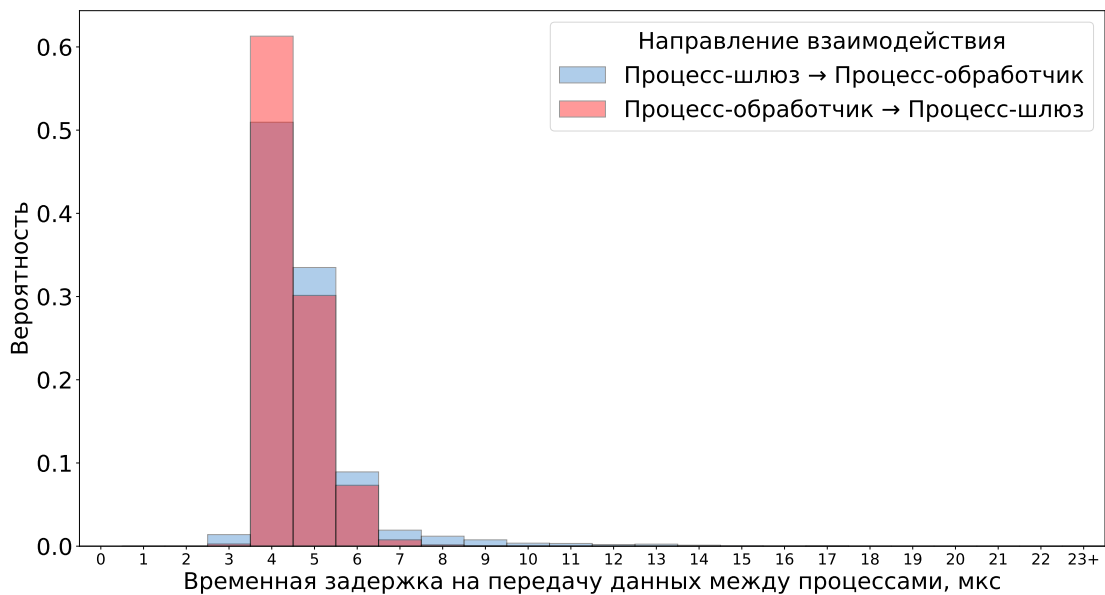


Рисунок 10 – Гистограмма временной задержки на передачу данных между процессами для метода, использующего разделяемую память для передачи данных, неблокирующий мультиплексор в разделяемой памяти и модель ”Лидер/Последователи“ при обслуживании заявок

ки. Исходя из разницы временной задержки на передачу данных в неблокирующем и блокирующем методах, 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 $\frac{24640.5 \text{ машинных циклов}}{2.1 \cdot 10^9 \text{ Гц}} \approx 11 \text{ мкс}$ 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, выполняется 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.

TBD: объяснить разброс в интервалах между заявками.

Выводы по главе 3

TBD: Вывод по очередной главе

ЗАКЛЮЧЕНИЕ

В данном разделе размещается 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. — URL: <https://kmu.itmo.ru/digests/article/3100> (дата обр. 02.05.2020).
- 2 Олеговна А. К. Исследование и разработка методов трассировки событий в параллельных и распределенных системах : Пояснительная записка выпускной квалификационной работы / Олеговна Анастасия Кузичкина. — Факультет ПИиКТ : Университет ИТМО, 06.2017.
- 3 Comparing and Evaluating epoll, select, and poll Event Mechanisms / L. Gammo [et al.] // Proceedings of the 6th Annual Ottawa Linux Symposium. — 2004. — Jan.
- 4 Draft: Have you checked your IPC performance lately? [Электронный ресурс] / S. Smith [et al.]. — 2012. — URL: <https://www.semanticscholar.org/paper/Draft-Have-you-checked-your-IPC-performance-Smith-Madhavapeddy/1cb3b82e2ef6a95b576573b8af0f3ec6f7bc21d9> (visited on 03/25/2020).
- 5 Gaztanaga I. Chapter 18. Boost.Interprocess [Электронный ресурс]. — URL: https://www.boost.org/doc/libs/1_63_0/doc/html/interprocess.html (visited on 03/25/2020).
- 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> (visited on 03/28/2020).

- 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/> (visited on 03/25/2020).
- 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 *Schmidt D. C.* Acceptor and connector // Pattern Languages of Program Design. — 1996. — Vol. 3. — P. 191–229.
- 12 *Schmidt D. C., Cranor C. D.* Half-Sync/Half-Async // Second Pattern Languages of Programs, Monticello, Illinois. — 1995.
- 13 *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.
- 14 The ADAPTIVE Communication Environment (ACE) [Электронный ресурс]. — URL: <https://www.dre.vanderbilt.edu/~schmidt/ACE.html> (visited on 02/14/2020).
- 15 *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>.
- 16 *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](https://doi.org/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 в мультиплексоре событий

```
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

```
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  
    номера 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();  
    }  
}
```