

**Федеральное государственное автономное образовательное  
учреждение высшего образования  
«Московский физико-технический институт  
(национальный исследовательский университет)»**

**УТВЕРЖДЕНО**

**Директор высшей школы  
программной инженерии  
А.В. Малеев**

	<b>Рабочая программа дисциплины (модуля)</b>
<b>по дисциплине:</b>	Теория и практика многопоточной синхронизации
<b>по направлению:</b>	Программная инженерия
<b>профиль подготовки:</b>	Разработка программно-информационных систем высшая школа программной инженерии высшая школа программной инженерии МФТИ - Яндекс
<b>курс:</b>	2
<b>квалификация:</b>	бакалавр

Семестр, формы промежуточной аттестации: 4 (весенний) - Дифференцированный зачет

Аудиторных часов: 60 всего, в том числе:

лекции: 30 час.

семинары: 0 час.

лабораторные занятия: 30 час.

Самостоятельная работа: 48 час.

Всего часов: 108, всего зач. ед.: 3

Количество контрольных работ, заданий: 2

Программу составил: А.В. Созыкин, канд. техн. наук, доцент

Программа обсуждена на заседании высшей школы программной инженерии МФТИ - Яндекс 28.04.2023

## Аннотация

В курсе изучаются основы многопоточного программирования и задачи, связанные с ним. Рассматриваются классические сценарии и механизмы синхронизации в многопоточных системах, такие как, например, мьютекс, условная переменная и барьер. Обсуждаются задачи и механизмы разных уровней, которые возникают в языках программирования, поддерживающих возможность асинхронного программирования, на примере языков программирования C++ и Golang. Немаловажное значение в курсе придаётся моделям памяти в различных языках программирования и архитектурах процессоров. Кроме этого, исследуются механизмы и стратегии garbage collection в различных языках программирования и проблемы, связанные с многопоточностью.

### 1. Цели и задачи

#### Цель дисциплины

- сформировать у студента понимание основных сценариев и инструментов многопоточной синхронизации;
- дать теоретические и практические знания о механизмах асинхронного программирования и асинхронного исполнения в различных языках программирования;
- дать студентам полную картину исполнения многопоточных систем: от моделей памяти процессора до устройства высокоуровневых примитивов как, например, futures.

#### Задачи дисциплины

- научить формулировать задачи в терминах изученных теорий, выбирать подходящий алгоритм для поставленной задачи;
- научить разрабатывать комбинации алгоритмов для решения поставленных задач, оценивать сложности алгоритмов, их модификаций и комбинаций, выбирать подходящие структуры данных для поставленных задач, реализовывать алгоритмы в обобщенной форме на языке программирования C++

### 2. Перечень формируемых компетенций

Освоение дисциплины направлено на формирование следующих компетенций:

Код и наименование компетенции	Индикаторы достижения компетенции
УК-2 Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	УК-2.1 Формулирует совокупность взаимосвязанных задач в рамках поставленной цели работы, обеспечивающих ее достижение. Определяет ожидаемые результаты решения поставленных задач
ОПК-2 Способен понимать принципы работы современных информационных технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности	ОПК-2.1 Способен использовать информационные технологии и программные средства, в том числе отечественного производства, при решении задач профессиональной деятельности
ОПК-4 Способен участвовать в разработке стандартов, норм и правил, а также технической документации, связанной с профессиональной деятельностью	ОПК-4.1 Знает основные правила оформления технической документации на различных стадиях жизненного цикла информационной системы
	ОПК-4.2 Владеет на практике методологией составления научно-технических отчетов (проектов)
ОПК-8 Способен осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий	ОПК-8.3 Умеет оптимизировать и проводить рефакторинг существующего кода для улучшения производительности и поддержки
	ПК-5.1 Умеет описывать архитектуру, устройство и функционирование информационных систем

ПК-5 Способен проектировать, разрабатывать, внедрять, сопровождать и снимать с эксплуатации информационные системы	ПК-5.2 Умеет определять оптимальные методы и инструменты разработки, внедрения и адаптации прикладного программного обеспечения информационных систем
	ПК-5.4 Имеет практический опыт разработки, внедрения и адаптации прикладного программного обеспечения информационных систем в различных программных средах
	ПК-5.3 Умеет разрабатывать прототип информационных систем в соответствии с требованиями и проводить его тестирование для проверки корректности архитектурных решений

### 3. Перечень планируемых результатов обучения по дисциплине (модулю)

В результате освоения дисциплины обучающиеся должны

знать:

- знать основные модели памяти, используемые в языках программирования и различных архитектурах процессоров, уметь описывать различия этих моделей памяти.

уметь:

- уметь описать основные сценарии многопоточной синхронизации и примитивы синхронизации, позволяющие эти сценарии воспроизвести;
- уметь описывать и реализовывать такие объекты асинхронного программирования, как *fibers*, *stackless coroutines*, *futures* и *promises*.

владеть:

- обладать навыками разработки и отладки многопоточных приложений, в том числе с использованием таких средств отладки, как *AddressSanitizer* и *ThreadSanitizer*.

### 4. Содержание дисциплины (модуля), структурированное по темам (разделам) с указанием отведенного на них количества академических часов и видов учебных занятий

#### 4.1. Разделы дисциплины (модуля) и трудоемкости по видам учебных занятий

№	Тема (раздел) дисциплины	Трудоемкость по видам учебных занятий, включая самостоятельную работу, час.			
		Лекции	Семинары	Лаборат. работы	Самост. работа
1	Проблемы синхронизации	10		10	12
2	Модель памяти, кэш	6		6	12
3	Lock-free структуры	8		8	12
4	События, задачи, пул потоков	6		6	12
Итого часов		30		30	48
Подготовка к экзамену		0 час.			
Общая трудоёмкость		108 час., 3 зач.ед.			

#### 4.2. Содержание дисциплины (модуля), структурированное по темам (разделам)

Семестр: 4 (Весенний)

##### 1. Проблемы синхронизации

Блокирующая синхронизация.

## Взаимное исключение

- Мотивация: примеры гонок, критические секции для группировки операций в атомарные блоки
- Постановка задачи взаимного исключения
- Гарантии консистентности (safety: mutual exclusion) и гарантии прогресса (liveness: deadlock freedom, starvation freedom)
- Атомарность и атомарная память
- Модель чередования для конкурентных исполнений
- Протоколы на чтениях/записях:
  - Мьютекс Петерсона для двух потоков
  - Обобщение на n потоков с помощью tournament tree
  - Свойство честности (fairness), doorway и wait-секции
  - Мьютекс Лампорта на временных метках, логические часы
  - Сравнение алгоритмов Петерсона и Лампорта
- RMW-операции: test-and-set, fetch-and-add,
- Протоколы на RMW-операциях:
  - TAS спинлок
  - Ticket спинлок
- Сравнение протоколов на атомарных RW-регистрах и RMW-регистрах
- Нижняя граница на число ячеек памяти для взаимного исключения
- Сплиттер и fast path для мьютекса Лампорта
- Дополнительно:
  - Задача renaming
  - Конструкция для ограниченных временных меток в алгоритме Лампорта

---

## Синхронизация: условные переменные и семафоры

- Паттерн коммуникации producer/consumer
- Блокирующая очередь, интерфейс, применение для пула потоков

## Условные переменные:

- Условные переменные как механизм ожидания и сигнализирувания
- Точная семантика операций
- Метафора с комнатами
- Предикат как функция от состояния, связь условной переменной и предиката
- Почему нужно будить в цикле:
  - Spurious wakeup
  - Intercepted wakeup
  - Loose predicate (на примере обобщенного семафора)
- Варианты сигнализирувания
- Инвариант и формальное док-во
- Сценарии с ошибками

Семафоры:

- Семафор как блокирующий счетчик
  - Интерфейс семафора, имена операций (P/V, acquire/release, wait/signal)
  - Семафор не пропускает сигналы, считает их
  - Сам счетчик недоступен в интерфейсе семафора
  - Семафор как автомат с жетонами
  - Реализация блокирующей очереди на семафорах, циркуляция жетонов
  - Реализация семафора на условных переменных
- 

Мелкогранулярные блокировки

- Задача обедающих философов и взаимные блокировки
- Wait-for граф для обнаружения взаимных блокировок, условия Коффмана
- Решение с помощью семафоров
- Решение с помощью нарушения симметрии
- Общая идея: упорядочивание локов, док-во корректности
- Главный пример - lock striping для хэширования цепочками
- Фиксированный массив мьютексов, страйпы
- Как делать расширение таблицы: отпустить лок страйпа и захватить все локи
- Локи захватываем справа налево, чтобы не возникло взаимной блокировки
- Гонки при расширении и как их избегать
- Нужно ли захватывать все локи, чтобы понять, что нас опередили?
- Проблема: Как узнать номер страйпа, не зная корзины? Как вычислить корзину, не захватив блокировки страйпа?
- Главный инвариант: каждый элемент не должен менять свой страйп при перехешировании
- Как добиться большего параллелизма в хэш-таблице: Reader/Writer блокировки
- Реализация с голоданием писателей
- Реализация с голоданием читателей
- Честная реализация
- Другой подход к построению словарей: скип-листы

## 2. Модель памяти, кэш

Как устроена память: когерентность кэшей и модель консистентности памяти

Когерентность кэшей и спинлоки

- Как работает с точки зрения синхронизации каждая отдельная ячейка памяти
- Кэши и иерархия памяти
- Проблема когерентности (синхронизации) кэшей
- Внутренняя реализация протоколов когерентности: message passing на шине памяти, свойства шины
- Протокол MSI для синхронизации, граф переходов
- Аналогия с R/W блокировками
- Оптимизация MSI -> MESI
- Дополнительно: MOESI, MESIF
- Когерентность и порядок модификаций: протокол когерентности упорядочивает записи в одну ячейку памяти
- Проблемы производительности, связанные с протоколом когерентности:
- cache ping-pong при захвате спинлока, TATAS-спинлок
- thundering herd при отпуске спинлока: спинлок Андерсона, CLH, MCS и другие реализации
- false sharing на примере распределенного счетчика

---

## Модель памяти

- Главные вопросы:
- Как упорядочиваются чтения и записи в разные ячейки?
- Что может вернуть конкретное чтение?
- Последовательная согласованность Лампорта
- Реальность:
- Примеры реордерингов для разных архитектур процессоров
- Реордеринги компиляторов
- Как гарантировать иллюзию последовательной согласованности, и при этом иметь реордеринги при исполнении?

## Последовательная согласованность для программ, свободных от гонок (SC-DRF):

- Гарантировать последовательную согласованность для любой программы дорого
- Процессору нужно делать реордеринги
- Программисту нужна простая модель исполнения
- Решение: обеспечим видимость последовательной согласованности только для корректно синхронизированных программ
- Модель памяти - контракт между разработчиками программ и разработчиками компиляторов/инженерами процессоров

## Главные идеи модели SC-DRF:

- Разделим ячейки памяти (и операции над ними) на два класса:
- Атомики - ячейки, которые используются для синхронизации, порядок обращения к ним и определяет синхронизацию (см. протокол Петерсона)
- Неатомарные ячейки
- Глобальный порядок для атомиков (synchronization order, SO)
- Отношение synchronizes-with (через SO)
- Отношение happens-before в модели отправки сообщений
- Happens-Before как транзитивное замыкание synchronization order и program order
- Видимость записей через happens-before для неатомарных операций
- Главная теорема: SO для атомиков + видимость записей через HB для остальных ячеек + DRF => последовательная согласованность

## Зачем такие сложности:

- Мы ослабили требования к упорядочиванию: последовательная согласованность только для DRF
- Взамен компилятор получает возможность делать больше реордерингов

## Классы реордерингов, которые допускает SC-DRF:

- Реордеринги между точками синхронизации на атомиках - порядок не могут наблюдать другие потоки
- Roach Motel
- Реализация модели памяти в языке программирования: барьеры памяти (memory barriers)
- Более слабые модели упорядочивания: acquire/release, relaxed

- Acquire/Release-семантика:
- Гарантируем только видимость через HB, глобального порядка на атомиках при этом нет
- Пример: Independent Reads of Independent Writes (IRIW)
- Relaxed-семантика
- Дополнительно: OoTA (out of thin air)

### 3. Lock-free структуры

Неблокирующая синхронизация, lock-free контейнеры, управление памятью

- Проблемы синхронизации: дедлоки, лайвлоки
- Прогресс в случае паузы потока, захватившего блокировку
- Идея неблокирующей синхронизации
- Строгие определения гарантий прогресса: lock-freedom, wait-freedom, obstruction freedom
- Не используем взаимное исключение
- CAS - швейцарский нож в мире RMW-операций
- Пример: как реализовать FAA с помощью CAS
- Общий механизм построения lock-free структур данных: ссылочные структуры и CAS-лупы
- Контейнеры:
- Стек Трайбера
- Проблемы: ABA, освобождение памяти
- Очередь Майкла-Скотта
- Список Харриса, применение для хэш-таблицы и скип-листа
- Проблемы с памятью:
- Освобождение памяти
- ABA при использовании пулов
- Подходы:
- Quiescent-based memory reclamation (QBMR)
- Hazard pointers
- Node recycling + tagged pointers

Дополнительно:

- Гарантия obstruction-freedom и снимки нескольких ячеек памяти

### 4. События, задачи, пул потоков

Линеаризуемость

- Мотивация: Что такое многопоточная FIFO-очередь?
- Более общие вопросы: как промоделировать конкурентные исполнения для структуры данных?
- Порядок:
- упорядочивание операций через критические секции
- на уровне ячеек памяти - последовательная согласованность для свободных от гонок программ
- Цель: получить модель чередования для высокоуровневых объектов

- Формальная модель: конкурентная и последовательная история, проекции на потоки и объекты

- Последовательная согласованность для высокоуровневых объектов

- Пример с реплицированным регистром - последовательная согласованность не позволяет комбинировать несколько объектов

- Линеаризуемость (атомарность)

- Док-во композируемости для линеаризуемости

- Точки линеаризации

- Пример с очередью Herlihy/Wing

- Линеаризация для ускорения структур данных: basket queue, elimination backoff stack

Дополнительно:

- Quiescent consistency и чудо-стек Шавита с призмами

- В случае операций над несколькими объектами (транзакции) нужна новая модель консистентности - сериализуемость

- Линеаризуемость и распределенные системы

- Local Linearizability?

---

Консенсус, wait-free иерархия

- Мотивация: Почему набор инструкций процессора именно такой? Какие операции лучше помогают синхронизировать потоки, а какие - хуже?

- Наводим интуицию

- Постановка задачи wait-free консенсуса, число консенсуса для операции/объекта

Как описать любой протокол консенсуса:

- Конфигурация, протокол как спуск по дереву конфигураций (вспомнить про линеаризуемость),

- бивалентные и унивалентные конфигурации, критическая конфигурация

- Универсальные свойства любого протокола: корень - бивалентный, любой протокол при спуске проходит через критическую конфигурацию

- Теорема: консенсус на атомарных R/W регистрах невозможен даже для двух потоков

- Пример с очередью (без док-ва): число консенсуса для wait-free очередей равно 2, а значит wait-free очередь нельзя реализовать только лишь на атомарных чтениях/записях

- Класс RMW-операций Common2: идемпотентные и коммутирующие операции

- Теорема: консенсус с помощью операций из Common2 невозможен даже для трех потоков

- Дополнительно:

- Робастность wait-free иерархии

- Рандомизация

- Универсальность консенсуса

---



## Универсальная конструкция

- Универсальность консенсуса

- Работаем со структурой данных как с черным ящиком, используем только последовательную спецификацию

Lock-free конструкция:

- Основная идея:

- Потоки разделяют не состояние структуры данных, а историю ее изменений (лог команд)

- Каждая операция состоит из двух шагов: 1) вставка элемента в лог команд, 2) применение команд из лога

поддерживаем общий лог команд (историю изменений)

- Реализация:

- Храним голову списка в виде массива

- Используем консенсус для вставки очередного узла в список

- Свобода от блокировок

- Линеаризация вставки

Wait-free конструкция:

- В чем проблема lock-free конструкции: поток может проигрывать вставки сколь угодно долго

- Идея: потоки должны синхронно помогать другим потокам вставлять узлы в лог команд

- Отличие от очереди Майкла-Скотта

- Реализация: анонсы вставок, round-robin по последовательным номерам узлов в истории

- Линеаризация: точка линеаризации операции может находиться в другом потоке!

- Замечания по эффективной реализации

- Управление памятью

- Параллели с распределенными системами - state machine replication

## 5. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)

Учебная аудитория, оснащенная компьютерами для каждого студента.

## 6. Перечень рекомендуемой литературы

### Основная литература

1. 19 смертных грехов, угрожающих безопасности программ. Как не допустить типичных ошибок [Электронный ресурс]. Электрон. версия печ. публикации / М. Ховард, Д. Лебланк, Д. Виега ; предисл. Амита Йорана. — М., ДМК Пресс, 2009

2. Защищенный код для Windows Vista, Электрон. версия печ. публикации / М. Ховард, Д. Лебланк. — Москва, Русская редакция, 2008

Литература, рекомендованная для самостоятельного изучения:

1. Maurice Herlihy, Nir Shavit «The Art of Multiprocessor Programming»

2. Anthony Williams «C++ Concurrency in Action»

### Дополнительная литература

Литература, рекомендованная для самостоятельного изучения:

1. Nir Shavit «Data structures in the multicore age»

2. Hans-J. Boehm, Sarita V. Adve «Foundations of the C++ Concurrency Memory Model»

## **7. Перечень ресурсов информационно-телекоммуникационной сети "Интернет", необходимых для освоения дисциплины (модуля)**

1. [http://neerc.ifmo.ru/wiki/index.php?title=Дискретная\\_математика,\\_алгоритмы\\_и\\_структуры\\_данных](http://neerc.ifmo.ru/wiki/index.php?title=Дискретная_математика,_алгоритмы_и_структуры_данных) – «Викиконспекты», сайт Санкт-Петербургского Института теоретической механики и оптики.

## **8. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень необходимого программного обеспечения и информационных справочных систем (при необходимости)**

- Документация к библиотеке Asio языка программирования C++  
<https://think-async.com/Asio/Documentation.html>
- Блогпост о Futures и Promises: <http://dist-prog-book.com/chapter/2/futures.html>
- Kotlin Coroutines Proposal: <https://github.com/Kotlin/KEEP/blob/master/proposals/coroutines.md>
- C++ Coroutines: Understanding operator co\_await  
<https://lewissbaker.github.io/2017/11/17/understanding-operator-co-await>
- Zero-cost futures in Rust: <http://aturon.github.io/blog/2016/08/11/futures/>
- Заметки о модели памяти ядра Linux:  
<https://github.com/torvalds/linux/tree/master/tools/memory-model/Documentation>
- CLion: среда разработки и отладки программ на языке C++

## **9. Методические указания для обучающихся по освоению дисциплины (модуля)**

Лабораторные работы и выполнение домашнего задания является одной общей работой студента. На занятиях особое внимание обращается на современные способы разработки программных продуктов. Необходимо требовать от студентов качественного и понятного программного кода наряду и в равной степени с правильной работой программы. На занятиях стоит постоянно разбирать ошибки, которые допускали студенты в своих программах, а также логические ошибки.

**ОЦЕНОЧНЫЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)**

<b>по направлению:</b>	Программная инженерия
<b>профиль подготовки:</b>	Разработка программно-информационных систем высшая школа программной инженерии МФТИ - Яндекс высшая школа программной инженерии
<b>курс:</b>	2
<b>квалификация:</b>	бакалавр

Семестр, формы промежуточной аттестации: 4 (весенний) - Дифференцированный зачет

**Разработчик:** А.В. Созыкин, канд. техн. наук, доцент

## 1. Компетенции, формируемые в процессе изучения дисциплины

Код и наименование компетенции	Индикаторы достижения компетенции
УК-2 Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	УК-2.1 Формулирует совокупность взаимосвязанных задач в рамках поставленной цели работы, обеспечивающих ее достижение. Определяет ожидаемые результаты решения поставленных задач
ОПК-2 Способен понимать принципы работы современных информационных технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности	ОПК-2.1 Способен использовать информационные технологии и программные средства, в том числе отечественного производства, при решении задач профессиональной деятельности
ОПК-4 Способен участвовать в разработке стандартов, норм и правил, а также технической документации, связанной с профессиональной деятельностью	ОПК-4.1 Знает основные правила оформления технической документации на различных стадиях жизненного цикла информационной системы
	ОПК-4.2 Владеет на практике методологией составления научно-технических отчетов (проектов)
ОПК-8 Способен осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий	ОПК-8.3 Умеет оптимизировать и проводить рефакторинг существующего кода для улучшения производительности и поддержки
ПК-5 Способен проектировать, разрабатывать, внедрять, сопровождать и снимать с эксплуатации информационные системы	ПК-5.1 Умеет описывать архитектуру, устройство и функционирование информационных систем
	ПК-5.2 Умеет определять оптимальные методы и инструменты разработки, внедрения и адаптации прикладного программного обеспечения информационных систем
	ПК-5.4 Имеет практический опыт разработки, внедрения и адаптации прикладного программного обеспечения информационных систем в различных программных средах
	ПК-5.3 Умеет разрабатывать прототип информационных систем в соответствии с требованиями и проводить его тестирование для проверки корректности архитектурных решений

## 2. Показатели оценивания компетенций

В результате изучения дисциплины «Теория и практика многопоточной синхронизации» обучающийся должен:

### знать:

- знать основные модели памяти, используемые в языках программирования и различных архитектурах процессоров, уметь описывать различия этих моделей памяти.

### уметь:

- уметь описать основные сценарии многопоточной синхронизации и примитивы синхронизации, позволяющие эти сценарии воспроизвести;
- уметь описывать и реализовывать такие объекты асинхронного программирования, как fibers, stackless coroutines, futures и promises.

### владеть:

- обладать навыками разработки и отладки многопоточных приложений, в том числе с использованием таких средств отладки, как AddressSanitizer и ThreadSanitizer.

### 3. Перечень типовых (примерных) вопросов, заданий, тем для подготовки к текущему контролю

Примеры практических заданий в течение семестра:

Найдите в данном коде асинхронного клиент-серверного приложения ошибку, объясните её суть и исправьте.

Реализуйте класс `stackless` `coroutine`. Подробнее:

<https://gitlab.com/Lipovsky/tpcc-course-2020/-/tree/master/tasks/3-tinyfiber/coroutine>

Реализовать мьютекс

Реализовать синхронные сокеты для фиберов над фреймворком `asio`, с их помощью написать эхо-сервер

Реализовать фреймворк `executors + futures/promises`

### 4. Перечень типовых (примерных) вопросов и тем для проведения промежуточной аттестации обучающихся

1. Напишите на псевдокоде примитив синхронизации `read/write-lock` с приоритетом для читателей/писателей.
2. Мьютекс называется рекурсивным (`reentrant`, `recursive`), если поток, владеющий мьютексом, может захватить его (вызвать метод `lock`), повторно. Для освобождения рекурсивного мьютекса поток должен вызвать `unlock` столько раз, сколько был вызвал `lock`. Например, `TAS` спинлок не является рекурсивным: попытка повторного захвата спинлока тем же потоком приведет к вечному циклу с `exchange` внутри. Стандартный системный мьютекс также не является рекурсивным. Реализуйте рекурсивный мьютекс с помощью обычного нерекурсивного мьютекса и условной переменной.
3. Напишите класс `CountDownLatch`, который инициализируется значением `count` и поддерживает две операции: 1) `Down()` - уменьшить значение счетчика на единицу 2) `Await()` - дождаться с блокировкой потока пока значение счетчика не опустится до нуля
4. Напишите `ReadWriteLock` с приоритетом для читателей
5. Напишите класс `TicketMutex`, который выдает потокам блокировку в порядке вызовов метода `Lock`. Ожидание на мьютексе должно быть блокирующим.
6. Этапы разработки параллельных алгоритмов: масштабирование набора подзадач и распределение подзадач между процессорами.
7. Оценка максимально достижимого параллелизма. Закон Амдаля.
8. Краткий обзор возможностей библиотеки `Qt` для организации многопоточности в приложениях. Границы применимости `Qt` для эффективной реализации параллельных вычислений
9. Режимы передачи данных в операциях парного обмена. Буферизованный режим передачи данных, режим передачи по готовности.
10. Моделирование параллельных программ. Основные виды и особенности моделей.

Критерии оценивания

отлично

10 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач, реализованы оптимальные алгоритмы, код оформлен в едином удобочитаемом стиле

9 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач, реализованы оптимальные алгоритмы

8 Полностью и вовремя решены все задачи без ошибок. Продемонстрирован грамотный подход к решению задач

хорошо

- 7 Полностью решены все задачи. Допущены несущественные ошибки.
- 6 Полностью решено большинство задач. В некоторых задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.
- 5 Полностью решено две трети задач. В некоторых задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.

удовлетворительно

- 4 Полностью решено более половины задач. В остальных задачах допущены и не исправлены ошибки, либо некоторые задачи решены частично.
- 3 Полностью решено более половины задач.

неудовлетворительно

- 2 Решено менее половины задач.
- 1 Не решено ни одной задачи.

## **5. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности**

Обучающийся решает задачи, подтверждает корректность решения с помощью системы автоматического тестирования и впоследствии защищает их у семинариста. Процесс защиты заключается в демонстрации кода решения задачи и объяснения его работы. По ходу защиты семинарист может задавать дополнительные вопросы по смежным темам. В случае если студент не может объяснить работу программы или ответить на вопросы, студент отправляется на доработку с возможностью повторной защиты. Количество повторных защит регламентируется преподавателем (семинаристом).

Защита может выполняться удаленно с использованием электронной почты, систем видеосвязи, внешним репозиторием системы контроля версий и др.