Обзор курса "Параллельное программирование"

Для кого задуман этот курс

Курс сделан для студентов Новосибирского Государственного Университета Механико-Математического Факультета по направлению подготовки "Системное программирование". Курс входит в план обязательных к посещению для студентов 2-3 курса.

Ожидаемые знания

Предполагается, что слушатель данного курса знаком с программированием на умеренном уровне ("считать из файла N чисел, отсортировать, вывести только простые"), может уверенно программировать на языке Java (классы, интерфейсы, generics, основные коллекции стандартной библиотеки).

Значительно легче будет разобраться с курсом, если имеется базовая математическая подготовка (1 семестр высшей математики или математического анализа, умение формально доказывать "очевидные" утверждения; 1 семестр алгебры или математической логики, чтобы не бояться частичных порядков), если прослушаны основы дискретной математики (понятие ориентированного графа, знакомство с любым волновым алгоритмом на графах, полный и частичный перебор вариантов).

Как извлечь пользу

Прочитайте разделы ниже, приходите на лекции и практические занятия, своевременно выполняйте домашние задания и ответственно готовьтесь к экзамену. Не стесняйтесь задавать вопросы.

Если курс для вас не обязателен к посещению, то рекомендую пробежаться по всем слайдам, обращая внимание на секции "further reading". Пролистать рекомендованные книги тоже будет полезно. Держите в уме, что это вводный курс на один семестр с умышленно ограниченной нагрузкой, возможно на просторах интернета вы найдете более подходящие материалы.

Цель курса

Цель данного курса – дать представление о многопоточности (параллелизме, одновременности, конкурентности) с точки зрения теории (с теоремами и техникой доказательств разных свойств многопоточных программ) и с точки зрения практики (как пользоваться примитивами синхронизации, как выстраивать свои программы, как отлаживать и оценивать эффективность прикладного кода). Так как мы должны оставить время на другие, не менее важные, дисциплины, то в курс входит всего 14 лекций и 7 практических занятий, чуть меньше стандартной семестровой нагрузки. Практические задания выстроены так, чтобы помогать слушателям осваивать всё более сложный материал, а потому сроки сдачи задач строго привязаны к расписанию лекций. Сдавать самую первую домашнюю задачу в конце семестра бессмысленно и, с точки зрения усвоения материала, даже вредно.

Сама концепция многопоточности, по мнению многих опытных программистов, весьма сложна и поэтому не хотелось бы, чтобы из предыдущего абзаца вы сделали вывод, что этот курс будет "легкой прогулкой". Просто на обсуждение будет вынесено меньше материала, чем в классическом курсе Herlihy & Shavit[1] или прекрасных курсах Романа Елизарова[2], интенсивах от Computer Science Center[3]. Предполагая, что слушатель курса

[1] https://booksite.elsevier.com/9780123973375/

[2] https://youtu.be/wdvjdChFUgo?si=sf8L8savyxcueorv

[3] https://youtu.be/fhcyQ2wU7Hk?si=L2pjJrWpieM92cgC

раньше ничего не слышал о потоках (threads), мы будем постепенно осваивать всё более сложные абстракции.

План курса

Курс разбит на 3 блока.

Первый блок ориентирован на практику, слушатели курса должны "набить руку" в написании небольших программ по координации потоков. Одновременно (no pun intended) с этим мы познакомимся с техниками написания и структурирования многопоточного кода, инструментами поиска ошибок, нефункциональными свойствами программ – пропускной способностью (throughput), отзывчивостью (latency), поддерживаемостью (maintainability).

- В лекции 1 мы познакомимся с параллелизмом и конкурентностью, кратко обозначим отличие между потоками и процессами операционной системы, укажем на роль планировщика задач операционной системы. В этой же лекции увидим состояние гонки (race condition), его частные проявления в виде гонки по данным (data race) и состояния тупика (deadlock). Познакомимся с "удобными" для параллелизма задачами (embarrassingly parallel problems) и теми, которые требуют более сложной координации, что позволит нам оценить теоретический предел эффективности с помощью закона Амдала.

- Лекция 2 познакомит с базовыми примитивами координации потоков – критической секцией (mutex) и условной переменной (conditional variable), с которыми связано много степеней свободы (acquisition order, reentrancy, fairness), различных техник программирования (data locking, code locking, waiting on predicate condition) и потенциальных проблем (deadlock, priority inversion, lost signal, spurious wakeup).

- В лекции 3 мы расширим наш "лексикон" продвинутыми структурами управления группами потоков (monitor, barrier, latch, read-write lock, semaphore, thread pool) и связанными с ними свойствами программ (throughput-latency trade-off, thundering herd problem, load balancing).

- Лекция 4 посвящена неочевидным идеям и шаблонам, используемым при многопоточном программировании: отмене задач (cancellation/interruption), владению ресурсом (ownership), использованию потоково-локальных данных (thread-local storage).

- Завершается первый блок лекцией 5, в которой фокус внимания смещается с алгоритмов на прикладные аспекты – поговорим про документацию нетривиальных протоколов, использование (и разумное **не**использование) ООП техник, стресс-тестирование, рандомизированный поиск ошибок, эмпирическую оценку надежности программ, базовые техники формальной верификации программ.

Второй блок погружает слушателя в пучину "настоящей" многопоточности, в которой на смену "объяснениям на пальцах" приходят более строгие рассуждения, математические формализмы, рассмотрение свойств современных процессоров и их подсистемы памяти, обобщение этих результатов на языки программирования высокого уровня.

- Лекция 6 начинает рассматривать многопоточные объекты как сложные сущности, расположенные на линии времени в некотором частичном порядке и иногда представляет несколько линий времени для независимых объектов. В отличие от физической концепции времени, в нашем курсе все операции будут подчиняться "более обозримым" законам, причем мы будем рассматривать разные виды согласованности операций (sequential consistency, linearizability, quiescent consistency). Формализация поможет нам более строго определить свойство надежности программ (correctness/safety), условия прогресса (liveness), неразрывности операций (atomicity).

- Имея в своем математическом арсенале частичные порядки, формализацию многопоточных объектов и знания о разных видах консистентности, в лекции 7 мы формализуем гарантии прогресса (obstruction-free, lock-free, wait-free), введем "самый простой" многопоточный объект – булеву ячейку памяти (register) и построим на основе этого целую иерархию различных многопоточных примитивов (safe, regular, atomic register). Кульминацией теоретической части станет возможность построить снимок состояния N ячеек памяти без использования блокирующих операций (wait-free atomic snapshot). Определение консенсуса потоков и "характеристического числа" многопоточного объекта (consensus number) станет приятным дополнением к нашему пониманию мира многопоточности.

- В лекции 8 мы изучим разные атомарные read-modify-write операции и с их помощью построим простые примитивы синхронизации (spin lock, test-and-test-and-set lock, lock-free stack), познакомимся с различными видами очередей (single/multi producer, single/multi consumer), узнаем об ABA проблеме.

- Лекция 9 перенесет нас от вершин теоретических изысканий к самым глубинам устройства современных процессоров и их подсистемы памяти. Мы обсудим роль кеширования и репликации данных, рассмотрим один из модельных протоколов, который позволяет этого достигнуть (MESI cache coherence protocol). После этого мы познакомимся с несколькими возможными оптимизациями исполнения (store buffer, load buffer, invalidate queue), которые делают многопоточное исполнение контринтуитивным. Чтобы починить наше видение мира мы воспользуемся барьерами памяти (memory barriers) и формализацией свойств процессора (hardware memory model), узнаем о степенях сложности в моделях памяти (weak memory model), о способах эмпирически изучать поведение вычислительной системы в экстремальных случаях (litmus tests).

- К концу второго блока мы успеем посмотреть на многопоточность и как на математический конструкт со множеством степеней свободы, и как на сложную физическую систему, которая стала трудна для понимания и анализа в результате ряда инженерных компромиссов, направленных на достижение максимальной эффективности. Лекция 10 проиллюстрирует как различные языки программирования пытаются скрыть сложность многопоточности от прикладного программиста, на какие "сделки с совестью" приходится идти разработчикам языков высокого уровня, какие подводные камни они приберегают для неосмотрительного читателя спецификации языка.

Третий блок включает в себя набор разрозненных концепций, о которых, по мнению автора курса, важно знать современному образованному системному программисту. Автор курса прекрасно осведомлен, что существуют не менее важные и полезные темы, которые стоило бы изложить в данном блоке. Гибкость данной части позволяет автору немного углубиться в те области многопоточного программирования, в которых у него больше практического опыта, а также пригласить замечательных докладчиков для гостевых лекций.

План третьего блока на 2025 год.

- В лекции 11 обсуждаются классические темы продвинутых курсов по многопоточности: эффективные реализации мьютексов и очередей (CLH/MCS), различные backoff стратегии и их оптимизации (recently-arrived-thread, single LIFO cell in fair queue), более подробно разбираются техники балансировки нагрузки и кражи работы (work stealing), тонкости обоснования конечности протокола (termination condition), приводится одна из версий классификации "всех-всех" многопоточных проблем.

- Лекция 12 знакомит с особенностями планирования исполнения без участия ядра ОС (user-space scheduling). Мы чуть-чуть познакомимся с I/O (berkeley sockets, блокирующие и неблокирующие варианты I/O) и классическими способами организовать эффективное многопоточное исполнение в присутствии блокирующих операций (future chaining, callback hell, async-await). Желая упростить жизнь прикладного программиста, мы перейдем от классического multi-tasking решения (ForkJoinPool + submit task) к M:N модели (легковесные процессы, корутины, зеленые потоки). Рассмотрим вариант неинвазивной реализации (continuation passing style, state machine, stackless coroutines), которая может быть выполнена на уровне frontend-компилятора языка программирования и привносит специфические сложности в прикладную разработку программ (проблема "двух цветов"). Рассмотрим подходы, основанные на реализации внутри виртуальной машины (stackful coroutines), особенности работы с такими системами.

- Лекция 13 пытается ответить на вопрос: а как правильно проектировать расширяемые и эффективные многопоточные системы? Мы познакомимся с вариантами дизайна стандартной библиотеки многопоточного языка (Java park/unpark, Java synchronizer), с универсальными низкоуровневыми примитивами синхронизации уровня ядра (futex/wait-on-address), встроенными в язык системами поиска ошибок (race finders, concurrent modification checkers), специализированными средствами мониторинга (observability API, perf, h/w counters, strace/ptrace, BPF), средствами наведения "иерархии" среди потоков исполнения (structured concurrency).

- В лекции 14 мы попытаемся взглянуть на многоагентные системы за пределами многопоточного программирования. Рассмотрим языки и фреймворки (полу-) автоматического распараллеливания программ (OMP, TBB, Java parallel stream API), распределенных вычислений (MPI, Apache Hadoop, Apache Spark). Немного поговорим о сложности программирования распределенных систем (ненадежность канала связи, скорость доставки сообщений, инверсия порядка сообщений) и тех сложностях, которые возникают у разных узлов кластера при попытке "договориться" (consensus protocols).