1. Масштабованість паралельної системи

Масштабируемость - способность системы, сети или процесса справляться с увеличением рабочей нагрузки при добавлении ресурсов. Система называется масштабируемой, если она способна увеличивать производительность пропорционально дополнительным ресурсам. Масштабируемость можно оценить через отношение прироста производительности системы к приросту используемых ресурсов. Чем ближе это отношение к единице, тем лучше. Также под масштабируемостью понимается возможность наращивания дополнительных ресурсов без структурных изменений центрального узла системы.

В системе с плохой масштабируемостью добавление ресурсов приводит лишь к незначительному повышению производительности, а с некоторого «порогового» момента добавление ресурсов не даёт никакого полезного эффекта.

Вертикальное масштабирование — увеличение производительности каждого компонента системы с целью повышения общей производительности.

Горизонтальное масштабирование — разбиение системы на более мелкие структурные компоненты и разнесение их по отдельным физическим машинам (или их группам), и (или) увеличение количества серверов, параллельно выполняющих одну и ту же функцию.

1. Поняття процесу та потоку.

Потоки и процессы представляют из себя последовательность инструкций, которые должны выполняться в определенном порядке. Инструкции в отдельных потоках или процессах, однако, могут выполняться параллельно. Процессы существуют в операционной системе и соответствуют тому, что пользователи видят, как программы или приложения.

Процесс - совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие.

Поток выполнения (от англ. thread — нить) — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы.

Процессы – выполнение нескольких задач. Потоки – разделение работы процесса.

Каждый поток имеет собственный счетчик команд и стек.

ОС каждому процессу создаёт не менее одного потока. При создании потока или процесса, операционная система генерирует описатель потока (содержит идентификатор потока, данные о правах доступа и приоритете, о состоянии потока и прочее). В исходном состоянии поток находится в приостановленном состоянии. Момент выборки потока на выполнение осуществляется в соответствии с принятым в данной системе правилом предоставления процессорного времени и с учетом всех существующих в данный момент потоков и процессов.

1. Базові засоби мови програмування для підтримки багатонитковості.

Сучасні мови програмування, які підтримують багатопотоковість, містять стандартні бібліотеки для роботи з потоками. В ці бібліотеки включено набір функцій для створення, запуску, призупинення, знищення, тощо.

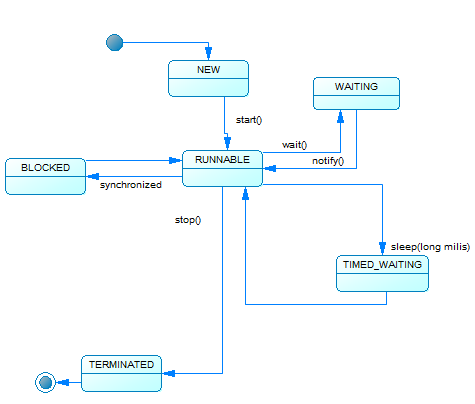
Підтримка багатонитковості.Важливою властивістю ОС є можливість розпаралелювання обчислень у межах одного завдання.Багатониткова ОС розподіляє процесорний час не між завдання­ми, а і між їхніми окремими галузями (нитками).

Багатопроцесорне оброблення.Загальноприйнятим є введен­ня в ОС функцій підтримки багатопроцесорного оброблення даних. Такі функції є в ОСSolaris 2.x фірми Sun, Open Server 3.x компанії Santa Crus Operations, OS/2 фірми IBM, Windows NT фірми Micro­soft і Netware 4.1 фірми Novell.

Багатопроцесорні ОС можна класифікувати за способом орга­нізації обчислювального процесу в системі з багатопроцесорною ар­хітектурою: асиметричні ОС і симетричні ОС.

1. Створення потоку. Вихід із потоку. Синхронізація при завершенні.

Существуют два способа создания и запуска потока: расширение класса **Thread** или реализация интерфейса **Runnable**. Переопределяется метод run



При выполнении программы объект класса Thread может быть в одном из четырех основных состояний: “новый”, “работоспособный”, “неработоспособный” и “пассивный”. При создании потока он получает состояние “новый” (NEW) и не выполняется. Для перевода потока из состояния “новый” в состояние “работоспособный” (RUNNABLE) следует выполнить метод start(), который вызывает метод run() – основной метод потока.

Получить значение состояния потока можно вызовом метода getState().

При роботі потоку ідентифікатор потоку залишається в несигнальному стані, після завершення роботи потоку, ідентифікатор переходить в сигнальний стан. Це дозволяє дізнатись чи завершився потік, або організувати очікування на завершення роботи кількох потоків.

для завершения потока необходимо освободить все занимаемые потоком ресурсы и выйти из метода run();

•для завершения другого потока его необходимо нотифицировать, вызвав метод interrupt(); при этом внутри прерываемого потока должна быть проверка запроса на прерывание (внутри цикла вызывать метод interrupted() или isInterrupted()), кроме того должно обрабатываться исключение InterruptedException, которое возникает, когда происходит попытка прервать поток, находящийся в состоянии ожидания.

[Більше дивись тут](http://crypto.pp.ua/2010/06/zhiznennyj-cikl-potoka-java/)

1. Синхронізація потоків.

Потоки иногда должны синхронизироваться. Один поток может ожидать результата другого потока, или одному потоку может понадобиться монопольный доступ к ресурсу, который используется другим потоком.

Все потоки одного процесса используют общие файлы, таймеры, устройства, одну и ту же область оперативной памяти, одно и то же адресное пространство. Это означает, что они разделяют одни и те же глобальные переменные.

Виртуальное адресное пространство процесса — это совокупность адресов, которыми может манипулировать программный модуль процесса. Операционная система отображает виртуальное адресное пространство процесса на отведенную процессу физическую память.

Один поток может использовать стек другого потока. Между потоками одного процесса нет полной защиты, потому что, во-первых, это невозможно, а во-вторых, не нужно. Чтобы организовать взаимодействие и обмен данными, потокам вовсе не требуется обращаться к ОС, им достаточно использовать общую память — один поток записывает данные, а другой читает их. С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга.

Операционная система обеспечивает каждый процесс отдельным виртуальным адресным пространством, так что ни один процесс не может получить прямого доступа к командам и данным другого процесса.

Если метод отмечен ключевым словом **synchronized,** то его или другие **synchronized-методы** для данного объекта может одновременно выполнить только один поток.

Если переменная объявлена как **volatile**, это означает, что ожидается её изменение несколькими потоками. JRE неявно обеспечивает синхронизацию при доступе к volatile переменным, но с одной очень большой оговоркой: чтение volatile переменных синхронизировано и запись в volatile переменные синхронизирована, а неатомарные операции – нет.

1. Атомарні операції.

**Атомарна операція** в [програмуванні](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) означає набір інструкцій з властивістю неперервності цілої операції. Атомарна операція виконується повністю (або відбувається відмова у виконанні), без переривань. Атомарність має особливе значення в [багатопроцесорних комп'ютерах](https://uk.wikipedia.org/w/index.php?title=%D0%91%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0&action=edit&redlink=1) і [багатозадачних](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%BD%D1%96%D1%81%D1%82%D1%8C) [операційних системах](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), оскільки доступ до ресурсів, що не розподіляються, повинен бути обов'язково атомарним.

Атомарна операція відкрита впливу тільки одної [ниті](https://uk.wikipedia.org/wiki/%D0%9D%D0%B8%D1%82%D1%8C).

Атомарність буває апаратна (коли безперервність забезпечується апаратурою) і програмною, коли використовуються спеціальні засоби міжпрограмної взаємодії ([м'ютекс](https://uk.wikipedia.org/wiki/%D0%9C%27%D1%8E%D1%82%D0%B5%D0%BA%D1%81" \o "М'ютекс), [семафор](https://uk.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%29)). За своєю суттю програмні засоби забезпечення атомарності складаються з двох етапів — блокування ресурсу і виконання самої операції. Блокування є атомарною операцією, яка або успішна, або повертає повідомлення про зайнятість.

Набір дій може вважатися атомним, коли виконуються дві умови:

1. Поки повний набір дій не завершується, ніякий інший процес не може знати про зроблені зміни (невидимість); і
2. Якщо будь-яка з дій не виконалася, тоді не виконується повний набір дій, і стан системи відновлюється до того стану, в якій це знаходилося перед тим, як будь-яка з дій почалася.
3. Монітори.

Мониторы представляют собой программные модули (объекты), которые реализуют (инкапсулируют) все необходимые действия с разделяемым ресурсом. Общий формат определения монитора может быть представлен следующим образом:

**Monitor <Name> {**

<объявления переменных>

<операторы инициализации>

<процедуры монитора>**}**

Как можно заметить, описание монитора достаточно близко совпадает с описанием класса в алгоритмическом языке С++. Принципиальное отличие состоит в том, что процедуры монитора, в обязательном порядке, выполняются в режиме взаимоисключения, т. е. при выполнении какой-либо процедуры монитора все остальные попытки вызова других процедур этого же монитора блокируются. Обеспечение такого правила выполнения процедур монитора должно осуществляться средой выполнения, в которой поддерживается концепция мониторов.

Помимо взаимоисключения процедур, другим важным свойством понятия монитора является его полная изолированность от остального кода программы:

Переменные монитора недоступны вне монитора и могут обрабатываться только процедурами монитора.

Вне монитора доступны только процедуры монитора.

Переменные, объявленные вне монитора, недоступны внутри монитора.

Подобная локализация (инкапсуляция) и объединение в рамках монитора всех критических секций потоков приводит к значительному снижению сложности логики параллельного выполнения.

1. Synchronized – методи та інструкція synchronized.

Очень часто возникает ситуация, когда несколько потоков, обращающихся к некоторому общему ресурсу, начинают мешать друг другу; более того, они могут повредить этот общий ресурс. Например, когда два потока записывают информацию в файл/объект/поток. Для предотвращения такой ситуации может использоваться ключевое слово synchronized.

Синхронизации не требуют только атомарные процессы по записи/чтению, не превышающие по объему 32 бит.

Если метод отмечен ключевым словом synchronized, то его или другие synchronized-методы для данного объекта может одновременно выполнить только один поток.

Синхронизировать объект можно не только при помощи методов с соответствующим модификатором, но и при помощи синхронизированного блока кода. В этом случае происходит блокировка объекта, указанного в инструкции synchronized, и он становится недоступным для других синхронизированных методов и блоков. Обычные методы на синхронизацию внимания не обращают, поэтому ответственность за грамотную блокировку объектов ложится на программиста.

1. Реалізація моніторів у Java.

Как известно, каждый объект в java имеет свой монитор, и потому, в отличие от того же C++, нет необходимости guard-ить доступ к объектам отдельными mutex-ами. Для достижения эффектов взаимного исключения и синхронизации потоков используют следующие операции:

1. monitorenter: захват монитора. В один момент времени монитором может владеть лишь один поток. Если на момент попытки захвата монитор занят, поток, пытающийся его захватить, будет ждать до тех пор, пока он не освободится. При этом, потоков в очереди может быть несколько.
2. monitorexit: освобождение монитора
3. wait: перемещение текущего потока в так называемый wait set монитора и ожидание того, как произойдёт notify. Выход из метода wait может оказаться и ложным. После того, как поток, владеющий монитором, сделал wait, монитором может завладеть любой другой поток.
4. notify(all): пробуждается один (или все) потоки, которые сейчас находятся в wait set монитора. Чтобы получить управление, пробуждённый поток должен успешно захватить монитор (monitorenter)

От того, есть ли contention на владение монитором, очень сильно зависит то, как производится его захват. Монитор может находиться в следующих состояниях:

* **init**: монитор только что создан, и пока никем не был захвачен
* **biased**: (умная оптимизация, появившаяся далеко не сразу) Монитор «зарезервирован» под первый поток, который его захватил. В дальнейшем для захвата этому потоку не нужны дорогие операции, и захват происходит очень быстро. Когда захват пытается произвести другой поток, либо монитор пере-резервируется для него (**rebias**), либо монитор переходит в состояние thin (**revoke bias**). Также есть дополнительные оптимизации, которые действуют сразу на все экземпляры класса объекта, монитор которого пытаются захватить (**bulk revoke/rebias**)
* **thin**: монитор пытаются захватить несколько потоков, но contention нет (т.е. они захватывают его не одновременно, либо с очень маленьким нахлёстом). В таком случае захват выполняется с помощью сравнительно дешёвых [CAS](http://en.wikipedia.org/wiki/Compare-and-swap). Если возникает contention, то монитор переходит в состояние inflated
* **fat/inflated**: синхронизация производится на уровне операционной системы. Поток паркуется и спит до тех пор, пока не настанет его очередь захватить монитор. Даже если забыть про стоимость смены контекста, то когда поток получит управление, зависит ещё и от системного шедулера, и потому времени может пройти существенно больше, чем хотелось бы. При исчезновении contention монитор может вернуться в состояние thin

1. Методи синхронізації wait, notify, notifyAll.

Методы wait() и notify() используются при освобождении и возврате блокировки в synchronized блоке.

Эти методы используются для управления потоками в ситуации, когда необходимо задать определенную последовательность действий без повторного запуска потоков.

* Метод wait(), вызванный внутри синхронизированного блока или метода, останавливает выполнение текущего потока и освобождает от блокировки захваченный объект. Возвратить блокировку объекта потоку можно вызовом метода notify() для конкретного потока или notifyAll() для всех потоков. Вызов может быть осуществлен только из другого потока, заблокировавшего, в свою очередь, указанный объект.