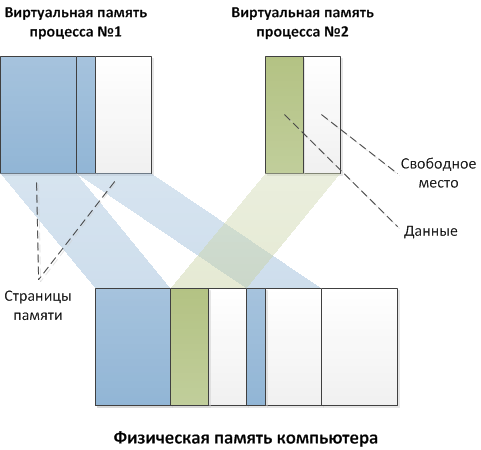
**Процесс** — это совокупность кода и данных, разделяющих общее виртуальное адресное пространство. Чаще всего одна программа состоит из одного процесса, но бывают и исключения (браузеры).

Процессы изолированы друг от друга, поэтому прямой доступ к памяти чужого процесса невозможен (взаимодействие между процессами осуществляется с помощью специальных средств).

Для каждого процесса ОС создает так называемое «виртуальное адресное пространство», к которому процесс имеет прямой доступ. Это пространство принадлежит процессу, содержит только его данные и находится в полном его распоряжении. Операционная система же отвечает за то, как виртуальное пространство процесса проецируется на физическую память.

Схема этого взаимодействия представлена на картинке. Операционная система оперирует так называемыми страницами памяти, которые представляют собой просто область определенного фиксированного размера. Если процессу становится недостаточно памяти, система выделяет ему дополнительные страницы из физической памяти. Страницы виртуальной памяти могут проецироваться на физическую память в произвольном порядке.



При запуске программы операционная система создает процесс, загружая в его адресное пространство код и данные программы, а затем запускает главный поток созданного процесса.

**Потоки**

Один поток – это одна единица исполнения кода. Каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса.

В системах с одноядерным процессором, система будет периодически переключаться между потоками, поочередно давая выполняться то одному, то другому потоку. Система запоминает состояние (контекст) каждого потока перед тем, как переключиться на другой поток, и восстанавливает его по возвращению к выполнению потока. В контекст потока входят такие параметры, как стек, набор значений регистров процессора, адрес исполняемой команды и прочее.

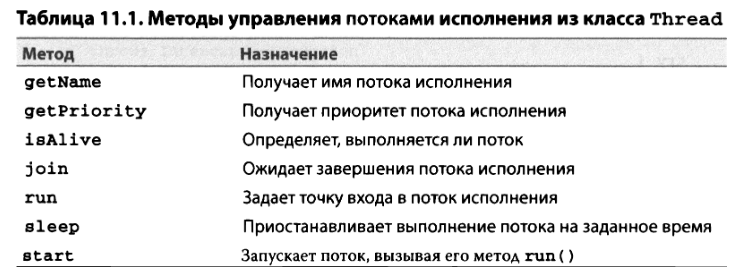
Связь между процессами, и переключение контекста ограничено и обходится дорого. Переключение между потоками дешевая, как и переключение контекста.

Потоки используются для обеспечения асинхронной работы всей исполняющей среды.

**Запуск потоков**

Каждый процесс имеет хотя бы один выполняющийся поток. Тот поток, с которого начинается выполнение программы, называется главным. В языке Java, после создания процесса, выполнение главного потока начинается с метода main(). Затем, по мере необходимости, запускаются другие, побочные потоки.

В Java поток представляется в виде объекта-потомка класса Thread. Этот класс инкапсулирует стандартные механизмы работы с потоком.



*Главный поток* создается автоматически, но им также можно управлять через объект класса Thread. Вызвав метод *currentThread()*, можно получить ссылку на тот поток, из которого он был вызван.

Запустить новый поток можно несколькими способами:

**Способ 1**

Создать объект класса Thread, передав ему в конструкторе нечто, реализующее интерфейс Runnable.



Этот интерфейс содержит метод *run()*, который будет выполняться в новом потоке. В run можно вызвать любые другие методы, объявлять переменные, как и в любом обычном методе. Когда метод run завершается, завершается и сам поток.

Поток запускается, при вызове метода *start()* класса Thread.

**Способ 2**

Расширить класс Thread, а затем получить экземпляр этого класса. В расширяющем классе должен быть объявлен метод *run()*, который является точкой входа в новый поток исполнения. Запуск потока также осуществляется методом start().

Второй способ следует применять, только если мы хотим переопределить какие-то методы класса Thread. В противном случае, лучше реализовывать интерфейс Runnable. Это также позволит нам наследоваться от каких-то других классов.

Чтобы определить, был ли другой поток завершен, есть 2 метода:

1. Вызвать для потока метод **isAlive()**, который возвращает true, если поток, для которого он был вызван, еще исполняется.
2. Метод **join()**, ожидает завершение того потока, для которого он был вызван. Можно указать промежуток времени, в течении которого требуется ожидать завершения потока.

**Приоритеты потоков**

Каждому потоку исполнения в java присваивается свой приоритет, который определяет поведение данного потока по отношению к другим потокам. Он определяется целым числом. Это число служит для принятия решения о переходе от одного потока к другому.

Чтобы установить приоритет потока, следует вызвать метод **setPriority()**



Уровни приоритета располагаются в промежутке от 1 (MIN\_PRIORITY) до 10(MAX\_PRIORITY). Значение по умолчанию 5(NORM\_PRIORITY).

Метод **GetPriority()** позволяет получить приоритет потока.

**Синхронизация**

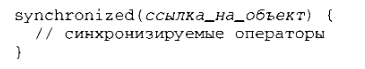
Если несколько потоков используют общие ресурсы, их нужно синхронизировать.

Монитор – объект, используемый в качестве взаимоисключающей блокировки. Только один поток исполнения может в одно и то же время владеть монитором. Можно рассматривать как контейнер, одновременно хранящий только один поток исполнения. Как только один поток войдет в монитор, другие потоки должны дождаться, пока он покинет монитор.

Для синхронизации кода используется ключевое слово **synchronized**.

У объектов имеются свои, неявно связанные с ними мониторы. Чтобы войти в монитор объекта, достаточно вызвать метод, объявленный с модификатором synchronized. Все другие потоки, пытающиеся войти в заблокированный монитор, будут приостановлены до тех пор, пока первый поток не выйдет из монитора. Чтобы выйти из монитора, владелец монитора просто возвращает управление из синхронизированного метода.

Синхронизировать можно не только метод, но и просто блок кода:



В скобках передается ссылка на синхронизируемый объект. Вызов метода, являющегося членом того же класса, на который указывает ссылка, произойдет только тогда, когда текущий поток исполнения войдет в монитор данного объекта.

Для организации взаимодействия потоков используются методы:

* *wait()* вынуждает вызывающий поток исполнения уступить монитор и перейти в состояние ожидания до тех пор, пока какой-нибуть другой поток не войдет в тот же монитор и не вызовет метод *notify().*
* *notify()* возобновляет исполнение потока, из которого был вызван метод *wait()* для того же самого объекта.
* *notifyAll()* возобновляет исполнение всех потоков, из которых был вызван метод *wait()* для того самого объекта. Одному из этих потоков предоставляется доступ.

Все эти методы объявлены в классе Object. Они могут быть вызваны только из синхронизированного контекста.

**Мьютекс, семафор, Монитор**

**Мьютекс** – специальный объект для синхронизации потоков. Обеспечивает взаимное исключение выполнения участков кодов. Имеет 2 состояния: свободен и занят.

Когда поток получает доступ к объекту, он переводит его в состояния занят. Другие потоки при этом будут ждать освобождения.

В java нет прямого доступа к мьютексу. Доступ к мьютексу есть только у java-машины.

**Монитор –** дополнительная надстройка над мьютексом. Монитор создает защитный механизм. Он срабатывает, когда поток входит в synchronized блок. Монитор в java по сути выражен с помощью слова synchronized.

**Семафор –** механизм синхронизации, который использует счетчик. Счетчик показывает, сколько потоков одновременно могут получать доступ к общему ресурсу.

**Типы ошибок синхронизации**

**Взаимная блокировка**

Проблема, когда потоки исполнения имеют циклическую зависимость от пары синхронизированных объектов. То есть, когда один поток ждет пока второй поток освободит ресурс. А второй, в свою очередь, ждет пока первый поток освободит ресурс. Потоков может быть и больше (допустим второй блокирует третий, а третий первый).

**Состояние гонки (Race condition)**

Ошибка программирования многозадачной системы, при которой работа системы зависит от того, в каком порядке выполняются потоки. Приведем пример. Пусть, один поток выполняет над общей переменной x операцию x = x + 3, а второй поток - операцию x = x + 5. Данные операции для каждого потока фактически разбиваются на три отдельных подоперации: считать x из памяти, увеличить x, записать x в память. В зависимости от взаимного порядка выполнения потоками подопераций финальное значение переменной x может быть больше исходного на 3, 5 или 8.

**Голодание потоков —** это ситуация, в которой поток не может получить доступ к общим ресурсам, потому что на эти ресурсы всегда претендуют какие-то другие потоки, которым отдаётся предпочтение.

**Состояния потока**

Поток исполнения может находится в нескольких состояниях. Чтобы получить состояние потока можно вызвать метод *getState()*. Это метод возвращает значения типа enum Thread.State.

Состояния потока:

* **BLOCKED –** поток приостановил выполнение, поскольку ожидает получения блокировки.
* **NEW –** поток создан, но не начал свое выполнение.
* **TERMINATED –** поток завершил выполнение.
* **TIMED\_WAITING –** поток приостановил выполнение на определенный промежуток времени, например после вызова метода sleep(). Поток переходит в это состояние и при вызове метода wait() или join().
* **WAITING –** Поток приостановил выполнение, поскольку он ожидает некоторого действия, например вызова версии метода wait() или join() без заданного времени ожидания.

