***11. Назначение планировщика и связь с синхронизацией.***

*Нити и Планировщик*

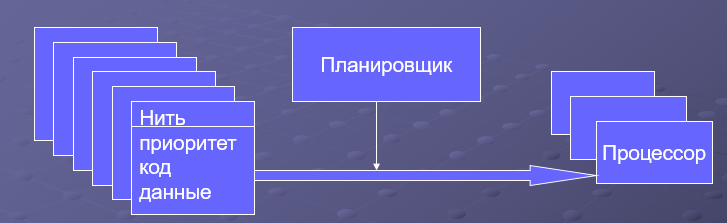
thread scheduler – планировщик java

[Про котенка и планировщик](https://habrahabr.ru/post/154609/) – в самом начале назначение планировщика

В Java есть Планировщик Потоков(Thread Scheduler), который контролирует все запущенные потоки во всех программах и решает, какие потоки должны быть запущены, и какая строка кода выполняться. Существует две характеристики потока, по которым планировщик идентифицирует процесс. Первая, более важная, это приоритет потока, другая, является-ли поток демоном(daemon flag). Простейшее правило планировщика, это если запущены только daemon потоки, то Java Virtual Machine (JVM) выгрузиться. Новые потоки наследуют приоритет и daemon flag от потока, который его создал. Планировщик определяет какой поток должен быть запущен, анализируя приоритет всех потоков. Потоку с наивысшим приоритетом позволяется выполниться раньше, нежели потокам с более низкими приоритетами.

Планировщик может быть двух видов: с преимуществом и без. Планировщик с преимуществом предоставляет определённый отрезок времени для всех потоков, которые запущены в системе. Планировщик решает, какой поток следующий запустится или возобновит работу через некоторый постоянный период времени. Когда поток запустится, через этот определённый промежуток времени, то выполняющийся поток будет приостановлен и следующий поток возобновит свою работу. Планировщик без приоритета решает, какой поток должен запуститься и выполняться до того, пока не закончит свою работу. Поток имеет полный контроль над системой настолько долго, сколько ему захочется. Метод yield() можно использовать для того чтобы принудить планировщик выполнить другой поток, который ожидает своей очереди. В зависимости от системы, на которой запущена Java, планировщик может быть либо с преимуществом, либо без него.

Планировщик определяет, какой поток должен запуститься, основываясь на номер приоритета, назначенный каждому потоку. Приоритет потока может принимать значения от 1 до 10. По умолчанию, значение приоритета для потока является Thread.NORM\_PRIORITY, которому соответствует значение 5. Так же доступны две других static переменных: Thread.MIN\_PRIORITY, значение 1, и Thread.MAX\_PRIORITY – 10. Метод getPriority() может использоваться для получения текущего значения приоритета соответствующего потока.



* Создание и разрушение нитей
* Распределение нагрузки между процессорами
* Приостановка и возобновление нитей
* Для синхронизации управляемого кода
* Для сборки мусора
* Для синхронизации неуправляемого кода и сборки мусора

*Синхронизация в Java*

[*смотреть всю главу 8*](http://metanit.com/java/tutorial/8.5.php) *– синхронизация на java с примерами.*

[*синхронизация java*](http://www.skipy.ru/technics/synchronization.html)

* *synchronized отображается в Monitor lock/unlock*
* *Методы Object отображаются в операции монитора*
* *Монитор*
* *lock – захват монитора, ожидание в случае конфликта*
* *unlock – освобождение захваченного монитора*
* *wait – освобождение монитора и ожидание извещения. Только для захваченного монитора*
* *notify – извещение одной ожидающей нити. Только для захваченного монитора*
* *notifyAll – извещение всех ожидающих нитей. Только для захваченного монитора*
* *При ожидании планировщик не планирует исполнение кода нити*
* *При извещении и освобождении монитора нить снова может быть запланирована для исполнения*