Текстовый вариант видеоурока из предыдущего шага

И мы, наконец, плавно подошли к реализации многозадачности в языке Go. Итак, Golang использует кооперативную многозадачность, о которой мы говорили.

Горутины – это нечто, похожее на треды операционной системы, только внутри Golang. Про них мы ещё поговорим. И как говорит известный Dr Gopher Strange: «Если я скажу тебе, в какой последовательности горутины будут выполнены, то этого не случится». И, на самом деле, отчасти, это – правда, потому что это всё вытекает из особенностей кооперативной многозадачности, плюс, особенностей планировщика задач в Go.

Что же за зверь такой эта горутина? Скорее всего, про горутины в языке Go вы много чего слышали и знаете. На самом деле, это очень удобный и классный инструмент, который облегчает жизнь разработчиков при реализации многопоточности в приложениях.

Простыми словами, горутина – это операция, которая может выполняться независимо от функции, в которой была запущена. И выполнение этой горутины может происходить параллельно. Для того, что в Go запустить какую-либо функцию в отдельной горутине – используется следующий синтаксис, приведенный на слайде. Сначала идет ключевое слово go и после него мы вызываем нужную нам функцию, которая как раз-таки будет запущена в отдельной гарутине.

И по традиции, давайте разберем горутины на примерах в коде. В данном файле я написал функцию workAndPrint, которая принимает в качестве параметра число, являющееся int-ом. Сюда мы, на самом деле, просто будем передавать порядковый номер запуска нашей функции. Внутри эта функция выводит информацию о том, что она стартанула и порядковый номер. Потом, мы

производим какую-то работу. Здесь, на самом деле неважно, что происходит. Для наглядности я просто в цикле 1000 раз перемножаю 2 числа, но это неважно, просто демонстрация того, что мы выполняем какую-то полезную работу для нас.

И в итоге, в конце в завершении функции мы выводим информацию о том, что эта функция завершилась. Также порядковый номер её выводим, чтобы отследить. И можем, например, вывести какие-то значения, которые получили в итоге.

Итак, в функции main мы в цикле вызываем нашу функцию workAndPrint 5 раз и передаём туда порядковый номер от 1 до 5. Давайте посмотрим, что произойдет после запуска этой программы.

Итак, как мы видим, мы по очереди вызываем нашу функцию workAndPrint. И она сначала стартует 1-ю функцию, потом пишет информацию, что JOB-а #1 закончилась и выводит какую-то…, какие-то значения. И дальше последовательно мы стартуем 2-ю, заканчиваем 2-ю, 3-ю, 4-ю и так далее до последней 5-й.

И что здесь происходит? Смотрите, в цикле я перебираю от 1 до 5 и вызываю нашу функцию workAndPrint. На самом деле, цикл здесь блокируется, и мы ждём, пока мы полностью не отработаем функцию workAndPrint. То есть пока функция workAndPrint работает, цикл дальше не двигается. Как только workAndPrint вернул какой-то результат и закончил свою работу, мы переходим к следующей итерации цикла.

Давайте, теперь запустим нашу последовательность вызовов функции workAndPrint параллельно, чтобы мы никак не блокировали этот наш цикл, а запускали эту функцию отдельно, не ждали, что она отработает и переходили сразу же к следующее итерации цикла. Как мы говорили в презентации – для этого мы можем воспользоваться как раз горутиной и нам достаточно здесь написать ключевое слово go. В этом случае при старте цикла мы в отдельной горутине

будем запускать наш метод workAndPrint и уже не ждать, пока она будет отрабатываться. Потому что с этого момента функция workAndPrint будет работать независимо от нашего цикла. И в принципе, от нашей функции main.

После этого сразу мы переходим к следующей итерации и так до конца цикла. И здесь есть такой интересный момент: я поставил здесь Sleep в 100 миллисекунд для того, чтобы наши горутины успели отработать. Дело в том, что, так как мы здесь не блокируемся, то наш цикл может пройти и вызвать все функции в горутине и быстро закончиться так и не дав отработать этим функциям. Чтобы такого не случилось, я здесь просто сделаю Sleep в 100 миллисекунд.

Давайте посмотрим теперь на вывод результата. Смотрите, как интересно: у нас теперь параллельно запускают функции workAndPrint независимо от последовательности в цикле for. Сначала, у нас стартует 4-я JOB-а, потом она заканчивается. Потом мы видим аж, 3 старта подряд: 5-й, 1-й и 3-й. Потом у нас заканчивается 5-я, стартует 2-я и так далее.

На самом деле, эти функции уже выполняются параллельно, просто вывод результата происходит в последовательном формате. И это как раз ещё показывает то, о чём мы говорили, что порядок выполнения горутин в Go не определен заранее, то есть они могут выполниться в разном порядке. Если я сейчас ещё раз запущу нашу программу, то мы получим совершенно другой результат. Как мы видим, здесь уже первым выполнилась JOB-а 1, потом 5, потом 3 и так далее. Как мы видим, в результате последовательность совершенно другая.

Давайте посмотрим, что произойдет, если я закомментирую нашу строчку со Sleep-ом. В этом случае мы просто в цикле вызовем наши горутины и сразу же завершимся. В результате мы ничего не наблюдаем. Я несколько раз вызываю

нашу программу и результат один и тот же. Почему так происходит? На самом деле, в Go функция main – это входная точка нашей программы, всё начинается с неё. И эта функция запускается в, так называемой, главной горутине. Все остальные горутины нашей программы будут вызваны из главной горутины, но либо из горутин, которые мы породили в главной горутине. Поэтому, как только главная горутина завершается, все остальные горутины тоже будут завершены.

И здесь мы просто банально не успеваем отработать наши горутины, которые висят и ждут, пока к ним переключим наше внимание. Поэтому, как только наша главная горутина завершается – всё остальное также схлопывается и завершается. И мы просто не успеваем это выполнять. И как мы говорили, когда рассматривали кооперативную многозадачность: переключения между задачами уже дело урок самих задач. То есть наши задачи как-то должны сообщать о том, что пора переключаться. За это в Go отвечает планировщик задач и, как раз-таки, этой строкой main Sleep мы говорим планировщику, что можно переключиться на другую горутину. То есть здесь главная горутина говорит, что я следующие 100 миллисекунд ничего не буду делать, а буду просто ожидать. Поэтому есть возможность переключиться на другую горутину и выполнить её.

Как только происходит sleep, мы уже переключаемся на другие горутины, которые могут выполняться уже параллельно. На самом деле, метод sleep – это не единственный вариант передать управление другой горутине. В языке Go много встроенных различных механизмов, которые сигнализируют планировщику о том, что пара переключить внимание на другую горутину. Один из таких – это как раз метод Sleep.

Вообще, метод Printf, который мы вызываем в workAndPrint – также способствует тому, чтобы переключиться на другую горутину и при его вызове может передаться управление в другую горутину. Но помимо этих встроенных триггеров, мы можем

самостоятельно это сделать руками, воспользовавшись встроенным пакетом runtime, в котором есть метод Gosched. Данный метод как раз явно говорит планировщику, что пора переключиться на другую горутину.

Давайте вызовем в нашем методе work and print этот метод и посмотрим на результат выполнения нашей программы. Смотрите, что здесь происходит: как мы видим, участились варианты того, что у нас START JOB происходит друг за другом. Давайте смотреть подробнее: у нас в цикле происходит запуск горутины workAndPrint. Потом мы из main горутины говорим, что пришло время переключиться на другую горутину, и мы переходим к выполнению нашей функции. Здесь мы печатаем START JOB, выполняем какую-то полезную нагрузку, и происходит переключение на другую горутину. То есть мы передаем управление другой горутине. Другая горутина – это тоже метод workAndPrint, который также выполняет START JOB, и доходит до конца только после того, как управление вернется к нему, потому что перед тем, как мы вводим END JOB, мы передаем управление. То есть получается, что у нас 4-я JOB-а стартует, доходит до передачи управления и передает управление 1-й JOB-е. 1-я JOB-а, в свою очередь, передаёт управление 2-й функции workAndPrint. И по-видимому, здесь у нас 2-я функция снова нас вернула на 4-ю, потому что здесь мы видим, вывод END JOB 4, то есть завершилась наша 4-я JOB-а. И так далее.