Текстовый вариант видеоурока из предыдущего шага

Определимся, зачем же всё-таки, в языке Go были созданы горутины. Смотрите, мы с вами вспомнили, что операционная система имеет в своём арсенале такие сущности, как «процесс» и «поток», – они управляются планировщиком операционной системы. Потоки позволяют разбивать нашу программу на части и выполнять её параллельно или асинхронно. Если бы мы в языке Go захотели выполнить что-то параллельно и асинхронно, то без горутин нам бы пришлось как-то взаимодействовать с планировщиком операционной системы и управлять потоками самой операционной системы.

Почему это не очень хорошо? – Потому что потоки сами по себе – это достаточно большая и сложная штука, поэтому разработчики языка решили оградить нас от потоков и создали такую абстракцию как горутина. При этом если вы пишете на Go, вы можете никогда не сталкиваться с потоками операционной системы, даже не знать об их существовании, потому что в Go вы работаете с горутинами, а с потоками уже Go справляется сам под капотом. То есть у Go есть свой планировщик задач, который разбирается с планировщиком задач операционной системы под капотом вашей программы.

И разработчики сделали так, что горутина получилась достаточно легковесной сущностью. В отличии от потока она очень легкая, например, горутина может быть пару килобайт, а поток – примерно пару мегабайт. Если мы запустим 1000 горутин – это будет в 1000 раз меньше по памяти, чем мы будем запускать 1000 тредов.

Собственно, это ещё даёт производительности в части переключения контекста. В одном потоке мы можем иметь множество горутин, которые работают между собой, и планировщик языка Go переключается между горутинами, которые легковесные. При этом у нас нет переключения контекста между разными потоками. В итоге у нас получается такая картина, что треды – это большие и

сложные сущности, а горутины – очень простые и быстрые. Получается, что у нас есть планировщик языка Go, который под капотом общается с планировщиком операционной системы. А разработчик, в свою очередь, знает только о существовании горутин и ни про какие треды он не думает, он оперирует только горутинами.

Чтобы это реализовать, у планировщика есть некоторый набор абстракций. Первая абстракция – это процессор, она обознается буквой «P». Процессор является некой абстракцией над ресурсом процессора нашего компьютера. Планировщик Go, на самом деле, запускает столько тредов, сколько у вас доступно ядер на вашем компьютере.

Следующая абстракция – это тред операционной системы. Он обознается буквой «M», потому что в Go эта абстракция называется Machine. И, наконец, асбатракция самой нашей горутины, которая запускается. В итоге мы получаем следующую схему взаимодействия с операционной системой, то есть у нас есть планировщик операционной системы. Дальше идут логические процессоры, ядра нашего процессора и, например, как приведено на слайде – планировщик языка Go на каждом логическом ядре нашего процессора запускает по 1 треду. И все горутины, которыми мы можем оперировать – уже запускаются в рамках этих 2 тредов и выполняются параллельно и асинхронно.

Чтобы поставить точку в теме с горутинами, давайте не забывать, что в языке Go реализована кооперативная многозадачность. Язык предоставляет удобный и легковесный инструмент как горутина. С помощью горутин язык исключил переключение контекстов в операционной системе, сведя этот процесс к минимуму, что даёт большой прирост производительности. И планировщик языка Go сам общается с планировщиком операционной системы, распределяет горутины по тредам, разработчику приходится работать только с горутинами.