Очередным важным инструментом для определения качества нашего кода является Race detector – это инструмент, позволяющий языку Go обнаружить участки кода, которые входят в состояние гонки. На самом деле, на ряду с профилировщиком Race detector можно отнести больше к некоторому (НРЗ 00:23) отладки и тестирования нашего кода. Но мы будем говорить об этом инструменте в разделе определения качества кода, так как в разделе тестирования мы сконцентрируемся немножко на другом.

Давайте на примере посмотрим, как мы можем использовать этот Detector. Здесь у меня есть фрагмент кода, в котором я искусственно создал состояние гонки. Что у нас тут происходит? У нас есть map-а data, в которой мы, например, регистрируем количество клиентов из какого-либо города. И, допустим, изначально у нас в Москве 100 клиентов, Екатеринбурге – 70. И как вы можете заметить, в 2-х разных goroutine-ах мы запускаем цикл, который обновляет наши значения по ключу. То есть мы инкрементируем значения для Москвы в отдельной goroutine-е и в другой goroutine-е также, которая запускается отдельно – инкрементируем значения по Екатеринбургу. После этого у нас идёт Sleep в одну секунду, и мы выводим, что дошли до конца нашей программы. Давайте запустим и посмотрим, что произойдёт.

Смотрите, на самом деле здесь мы сразу провалились в ошибку, и наш код даже не взлетел, потому что мы получили fatal error: concurrent map writes, то есть язык сразу определил, что мы здесь пытаемся записать в map-у из нескольких goroutine, и у нас возникает состояние гонки.

На самом деле, не всегда это состояние бывает настолько очевидным. В некоторых ситуациях наш код работает, и всё проходит гладко и незаметно, но в каких-то узких кейсах, которые стреляют не так часто, может возникнуть именно это состояние гонки. И тогда при запуске нашей программы мы об этом не узнаем,

программа запустится как обычно. В этом случае мы можем воспользоваться как раз-таки Race detector-ом, чтобы запустить нашу программу, скажем так, в отладочном режиме, которая даёт нам информацию про состояние гонки. Делается это с помощью такого флажка race при запуске go run. Давайте запустим и посмотрим, что произойдет.

Смотрите, в этом случае программа у нас не вылетела, и мы даже дошли до последней строчки, где мы выводим на экран «END MAIN». Но при этом, вывод нам говорит о том, что мы нашли Race condition-ы, то есть у нас в программе есть состояние гонки. Давайте проанализируем вывод.

Смотрите, здесь мы запускали без Race detector-а и, на самом деле, мы просто получили fatal error, а потом некий (HP3 04:02) – какие-то внутренности. После того, как мы воспользовались Race detector-ом, здесь мы уже получаем более конкретные данные. Здесь нам говорится о том, что произошло чтение и указывается, где конкретно. Допустим, здесь у нас произошло чтение данных.

До этого также мы осуществляли запись данных, но уже в другой goroutine-е. Таким образом ссылаясь на участки кода, в которых произошёл Race condition, мы можем их обработать и переписать для того, чтобы избежать этих ситуаций.