Текстовый вариант видеоурока из предыдущего шага

Друзья, всем привет. И добро пожаловать на очередную серию уроков, посвященную языку Golang. С вами снова я Тигран, и в этой части мы будем говорить о параллельности в Go.

Мы затронем такие темы, как теорию многозадачности, посмотрим на разницу между тредами и горутинами, разберем обработку ошибок, базовые принципы синхронизации между горутинами. Посмотрим, конечно же, на такие инструменты, как WaitGroup, типы каналов и Select-ы. В итоге, в конце данной части мы разберем пакет context, который тоже очень полезен на практике.

Многозадачность – это свойство среды выполнения наших программ, которая позволяет нам параллельно или псевдопараллельно обрабатывать задачи.

В теории многозадачности выделяют 3 основные подхода к её реализации. Называются они следующим образом: это простое переключение, кооперативная многозадачность и вытесняющая многозадачность.

При простом переключении задач система загружает в память несколько приложений, но при этом процессорное время отдаётся только основному. То есть образуется своего рода, очередь из наших задач, которые готовы к выполнению, и система просто забирает их по очереди. Но тут есть очевидный минус в том, что большие и долгие задачи занимают очередь и время для выполнения. При этом маленькие и более важные задачи простаивают. Данный метод не очень подходит для интерактивных систем, поэтому в современных системах самостоятельно он не используется. По крайней мере, если даже используется, то очень редко.

При кооперативной многозадачности картина немного отличается. Тут у нас кооперация задач. То есть предполагается, что задачи все такие вежливые по отношению к друг другу и чуткие, поэтому они самостоятельно уступают место в

нужный момент и говорят, что: «Я закончил делать что-то полезное и могу немного подождать, перед тем как сделать что-то ещё». Как мы видим на слайде, процессорное время распределяется между задачами таким образом, что Process A, допустим, в какой-то момент говорит, что: «Я что-то сделал и могу подождать», – при этом процессор понимает, что у него есть возможность переключиться на следующую задачу – он переключается на Process B. Process B таким же образом что-то выполняет. Доходит до момента, когда можно будет переключиться на следующий и сообщает процессору о том, что пора переключиться на другой процесс. В этот момент он опять переходит в режим ожидания, и выполнение переключается на Process A.

В свою очередь, в вытесняющей многозадачности происходит обработка задач по приоритизации. И система сама решает, когда и на какую задачу нужно переключиться. Тут уже сами задачи не могут ничего с этим поделать – они никак не влияют на этот процесс переключения. Система сама их принудительно переключит, например, после истечения какого-то кванта времени. Стоит сказать, что данный подход достаточно распространен и в большинстве современных операционных системах используется именно вытесняющая многозадачность. Такие операционные системы как Windows, Linux, Mac OS в той или иной степени реализуют именно вытесняющую многозадачность. Также данный подход очень часто используется в Real-Time системах.