Как устроен планировщик горутин в Go: G-M-P модель

Go использует уникальную модель управления конкурентностью — **G-M-P** (Goroutine, Machine, Processor), которая обеспечивает масштабируемость, высокую производительность и лёгкость работы с параллелизмом.

□□ Компоненты Go-планировщика

Обозначение	Название	Что это такое
G	Goroutine	Логическая единица выполнения (код + стек)
M	Machine (Thread)	Системный поток (OS thread)
P	Processor	Планировщик, управляющий выполнением G

□□ Как это работает?

- **M (Thread)** — физический поток, выполняющий код
- **P (Processor)** — абстрактный процессор: даёт M-у задачу (G)
- **G (Goroutine)** — логическая задача, которую нужно выполнить

> M нужен P, чтобы выполнять G. Без P поток (M) простаивает.

□□ Модель в действии

При запуске:

```
runtime.GOMAXPROCS(4)
```

Ты создаёшь 4 **P (процессора)**. Это означает, что Go может **одновременно** запускать до 4 горутин **в реальном параллелизме**.

□□ Визуально:



□ Work-stealing

Если один P (процессор) простаивает, он может “украсть” G из очереди другого P. Это делает систему ещё более гибкой и **минимизирует простои**.

□□ Примеры поведения

Сценарий	Как себя ведёт Go

1 млн горутин	Go спокойно справится
Ожидание по `time.Sleep`	P отходит от M и отдаёт другим
Блокировка I/O	M блокируется, но P берёт новый M
`runtime.GOMAXPROCS(n)`	Ограничивает кол-во одновременно исполняемых G

Вывод:

Go-планировщик:

- работает по **G-M-P** модели
- очень эффективно управляет тысячами горутин
- масштабируется лучше, чем системы, построенные на потоках ОС
- делает конкурентное программирование простым и надёжным

Revision #1
Created 2025-06-27 12:00:39 UTC by Admin
Updated 2025-06-27 12:03:36 UTC by Admin