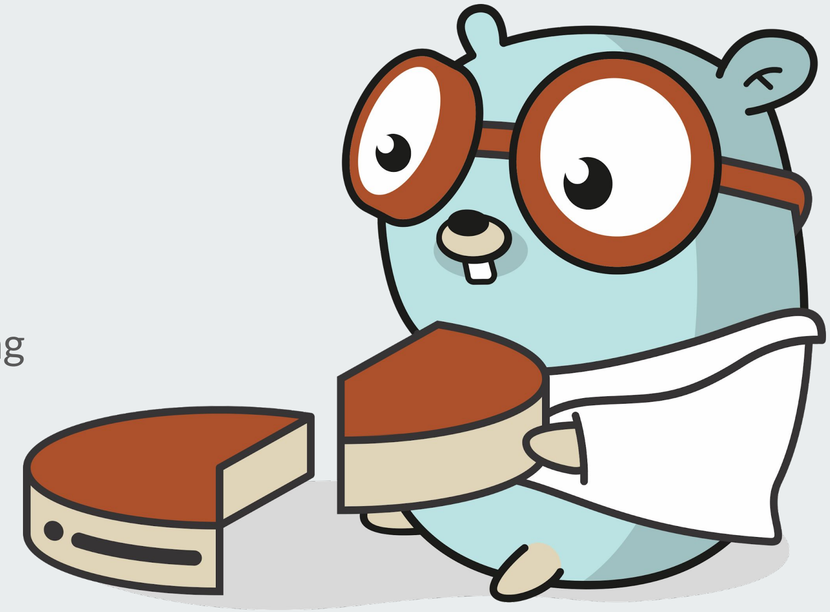
Goroutines demystified.

Internals of goroutines, threading & scheduling



Gufran Mirza
gufranmirza

Goroutines??

A goroutine is a lightweight thread managed by the Go runtime. They are called lightweight threads because they require less processing time

- **Smaller default stack size**
- **Lighter context switching** :
    - setup and teardown don’t require call to the kernel

User-threads

Ok, so basically they are just user threads : an implementation of threads and scheduling running on top of the OS.

- But wait how is this a good idea ?
- Why is it cool to reimplement something already provided by your kernel ?

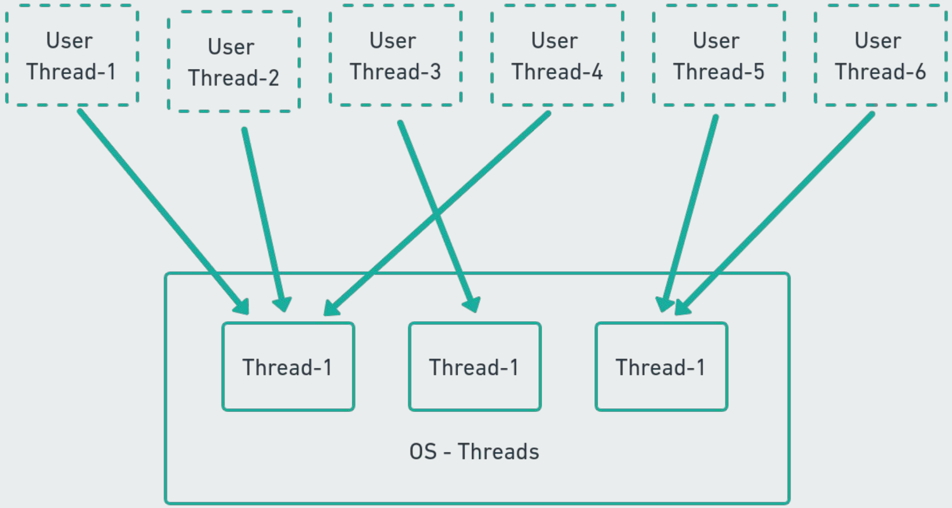
User-threads ideas

Some nice ideas to make profit of user threads :

- Allocate kernel threads when creating the first user threads.
- Park for reuse the kernel threads after the user thread ends.
- Schedule user threads to run on respecting kernel threads.
- Lightweight in terms of setup and teardown cost

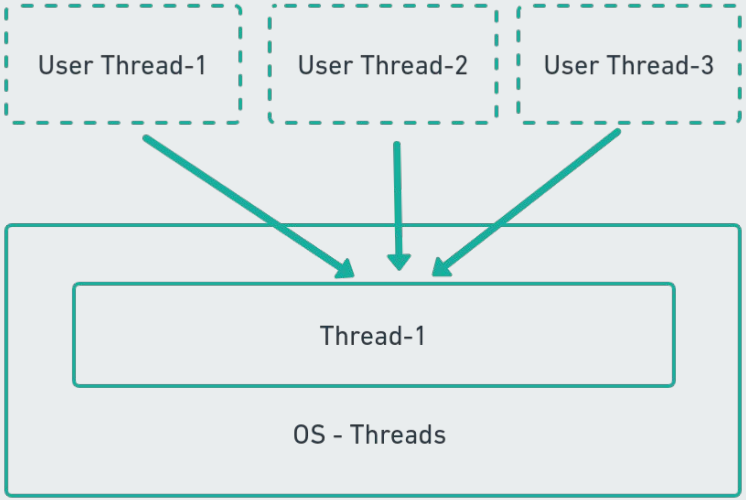
User-threads implementation

Multiplexing **low-cost user-threads** on **high-cost kernel threads** :



User-threads implementation

Multiple user thread can run on the same associated kernel thread.



Scheduling concepts

The scheduler manages a runqueue of runnable goroutines.

- When it wants to schedule a goroutine it pops it out of the runqueue and schedules it on a available kernel thread
- Instantiating kernel thread if needed and possible.

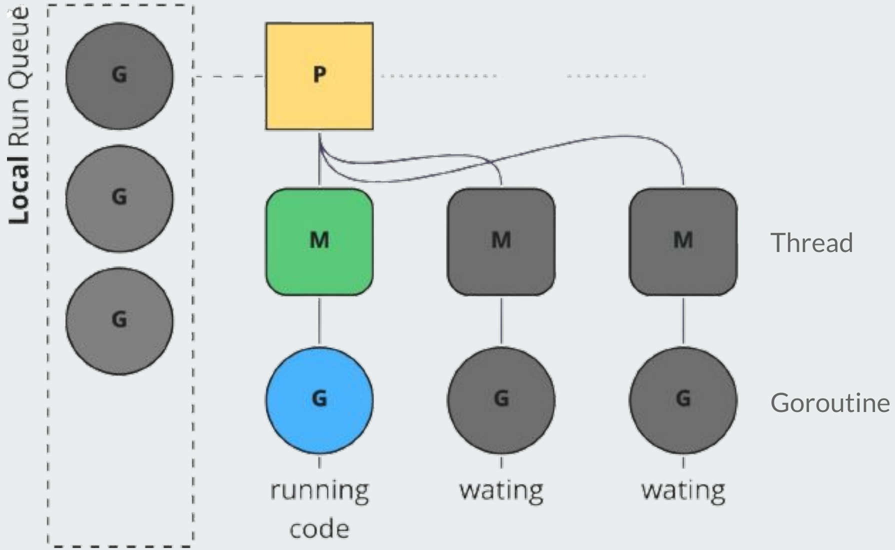
Scheduling concepts

Go implements theses concepts through 3 important structures in the runtime code :

The G struct (Goroutine)	The M struct (Kernel Thread)	The P struct (Linked List/Processor)
- Represents a runnable goroutine - Contains informations about its stack - Its current status and its associated P	- Represents a kernel thread - Contains two important pointers: - One to the currently running G and - Another one to its attached P.	- Represents a scheduling context - Contains a list of runnable Gs.

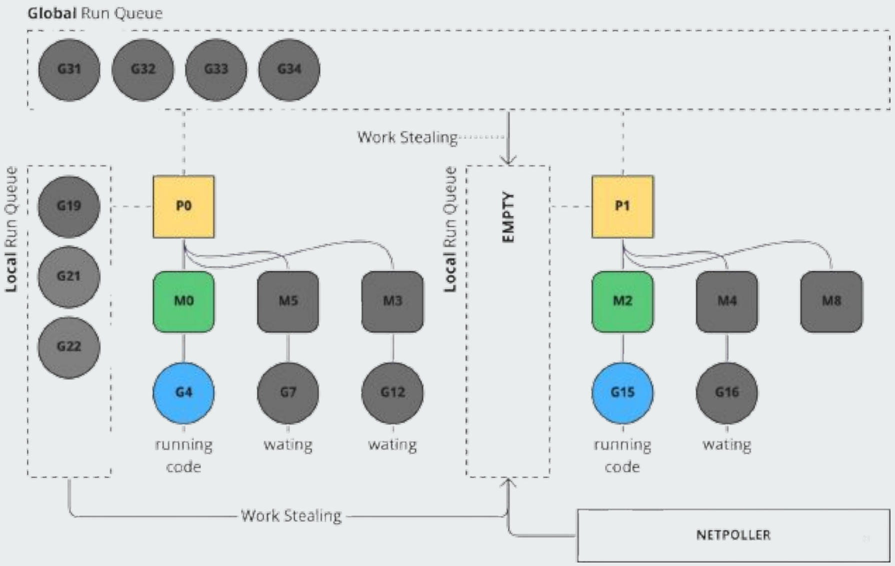
Go Scheduler

Its job is to distribute runnable goroutines (G) over multiple worker OS threads (M) that run on one or more processors (P). Processors are handling multiple threads



Goroutines queues

Go manages goroutines at two levels, local queues and global queues. Local queues are attached to each processor, while the global queue is common.



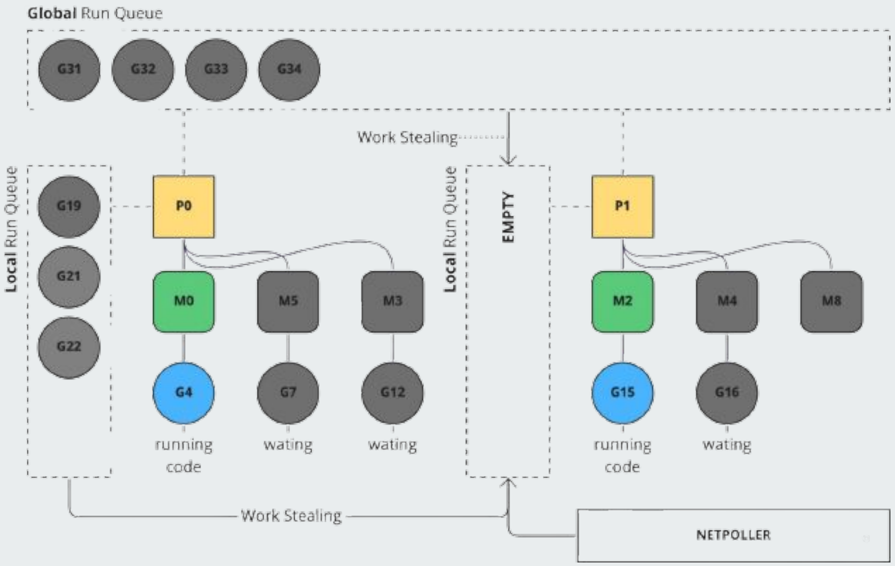
Work-stealing

When a processor does not have any work, it applies the following rules until one can be satisfied

- Pull work from the local queue
- Pull work from the global queue
- Pull work from network poller
- Steal work from the other P’s local queues

Work-stealing

P1 is looking for work. However, its local queue, the global queue has some G, and the network poller are empty. The last solution is to steal a job from global queue:



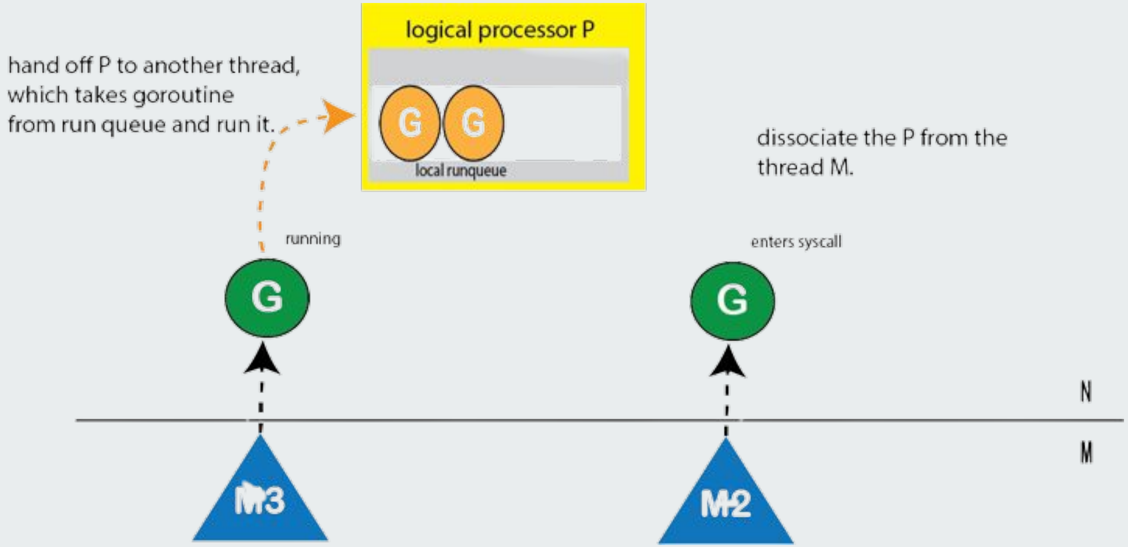
Blocking Calls

The Blocking SYSCALL method is encapsulated between **runtime.entersyscall(SB)** **runtime.exitsyscall(SB)**

- In a literal sense, some logic is executed before entering the system call, and some logic is executed after exiting the system call.
- This wrapper will automatically dissociate the P from the thread M when a blocking system call is made and allow another thread to run on it.

Blocking Hadoff

The way we can restore parallelism is that when we enter the system call, we can wake up another thread, which will select the runnable goroutine from the run queue.



Blocking Exit

What happens Once blocking syscall exits?

- Runtime tries to acquire the exact same P, and resume the execution.
- Runtime tries to acquire a P in the idle list and resume the execution.
- Runtime put the goroutine in the global queue and put the associated M back to the idle list.

Questions ?

Credits

- https://medium.com/a-journey-with-go/go-work-stealing-in-go-scheduler-d439231be64d
- https://betterprogramming.pub/deep-dive-into-concurrency-of-go-93002344d37b
- https://medium.com/@ankur_anand/illustrated-tales-of-go-runtime-scheduler-74809ef6d19b
- https://github.com/golang/go/wiki/Performance#scheduler-trace

Contact me

Gufran Mirza

Software engineer @IBM Cloud

@_imGufran

@gufranmirza