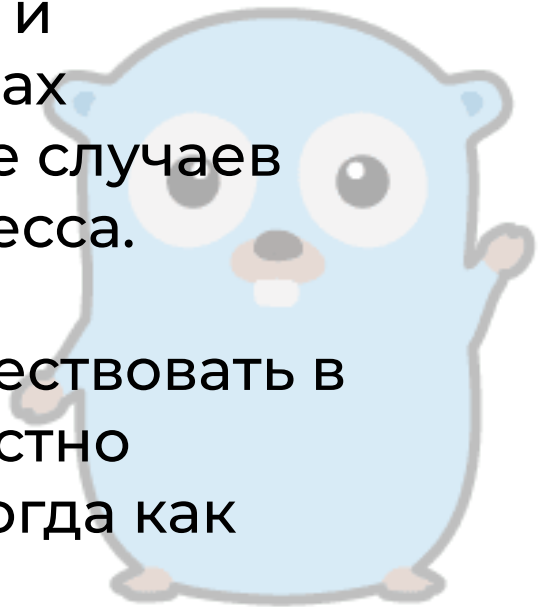
ЛЕКЦИЯ 6

Потоки. Горутины. Каналы

Потоки. Threads

Поток выполнения:

Наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса.



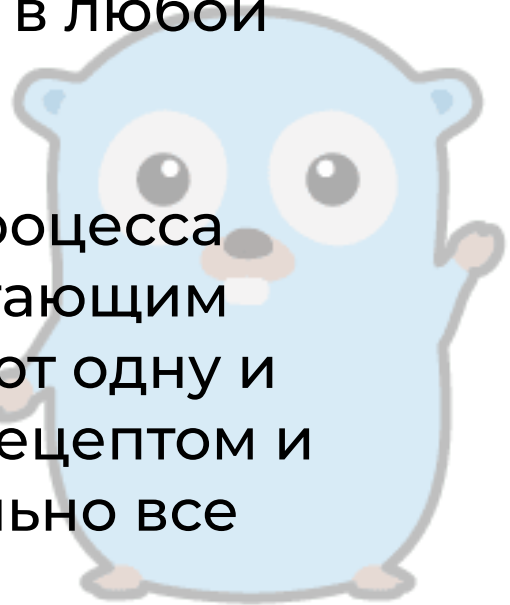
Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов.

Потоки. Threads

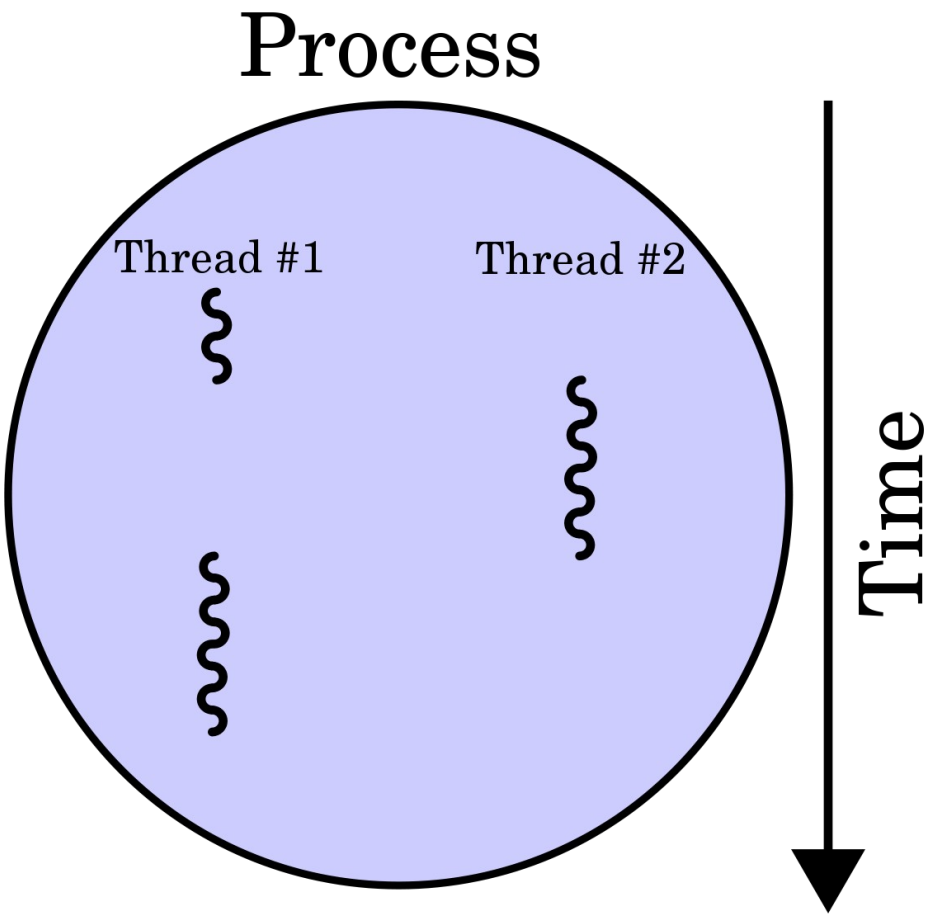
Поток выполнения:

В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени).

В качестве аналогии потоки выполнения процесса можно уподобить нескольким вместе работающим поварам. Все они готовят одно блюдо, читают одну и ту же кулинарную книгу с одним и тем же рецептом и следуют его указаниям, причём не обязательно все они читают на одной и той же странице.



Потоки. Threads

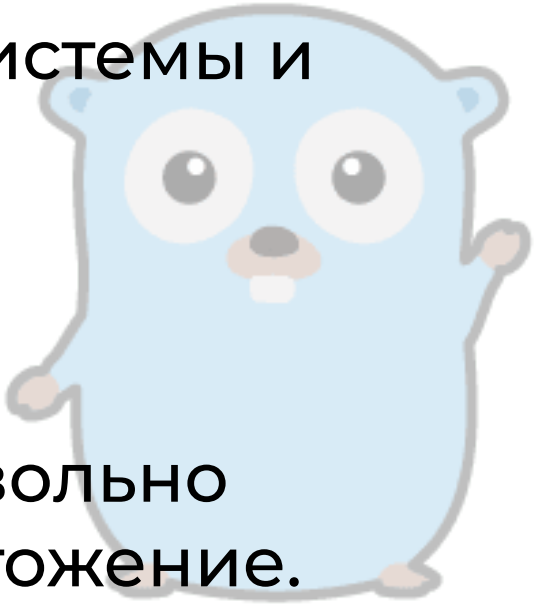


Потоки. Threads

Поток выполнения:

Потоки — это дорогостоящие объекты, которые занимают память, могут использовать различные ресурсы системы и находиться в разных состояниях.

Для их создания требуется время. В сравнении с процессами они менее ресурсоемки, но все же требуют довольно больших затрат на создание и уничтожение.

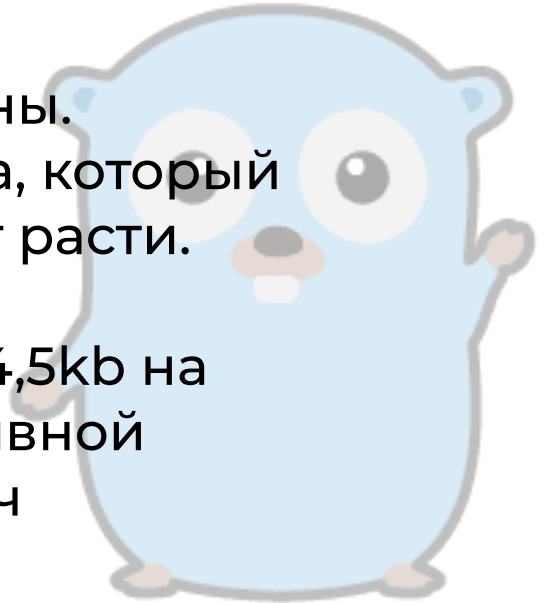


Горутины. Goroutines

Горутины:

Горутина (goroutine) — это функция, выполняющаяся конкурентно с другими горутинами.

Обратите внимание, горутины очень легковесны. Практически все расходы — это создание стека, который очень невелик, хотя при необходимости может расти.



В среднем можно рассчитывать примерно на 4,5kb на горутину. То есть, например, имея 4Gb оперативной памяти, вы сможете содержать около 800 тысяч работающих горутин.

Горутины. Goroutines

Горутины:

В исходном коде (src/pkg/runtime/proc.c) приняты такие термины:

G (Goroutine) — Горутина
M (Machine) — Машина

Горутины. Goroutines

Горутины:

Каждая Машина работает в отдельном потоке и способна выполнять только одну Горутину в момент времени. Планировщик операционной системы, в которой работает программа, переключает Машины.

Число работающих Машин ограничено переменной среды GOMAXPROCS или функцией runtime.GOMAXPROCS(n int). По умолчанию оно равно 1. Обычно имеет смысл сделать его равным числу ядер.

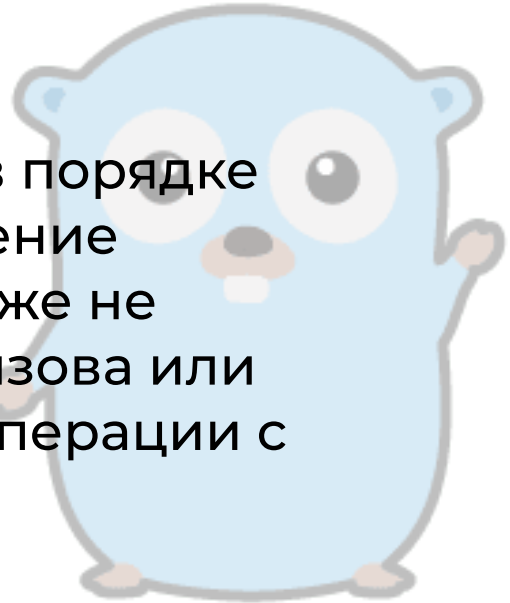
```
runtime.GOMAXPROCS(runtime.NumCPU())
```

Горутины. Goroutines

Горутины:

Цель планировщика (scheduler) в том, чтобы распределять готовые к выполнению горутины (G) по свободным машинам (M).

Готовые к исполнению горутины выполняются в порядке очереди, то есть FIFO (First In, First Out). Исполнение горутины прерывается только тогда, когда она уже не может выполняться: то есть из-за системного вызова или использования синхронизирующих объектов (операции с каналами, мьютексами и т.п.).



Горутины. Goroutines

Горутины:

Не существует никаких квантов времени на работу горутины, после выполнения которых она бы заново возвращалась в очередь.

Чтобы позволить планировщику сделать это, нужно самостоятельно вызвать runtime.Gosched().

Как только функция вновь готова к выполнению, она снова попадает в очередь.

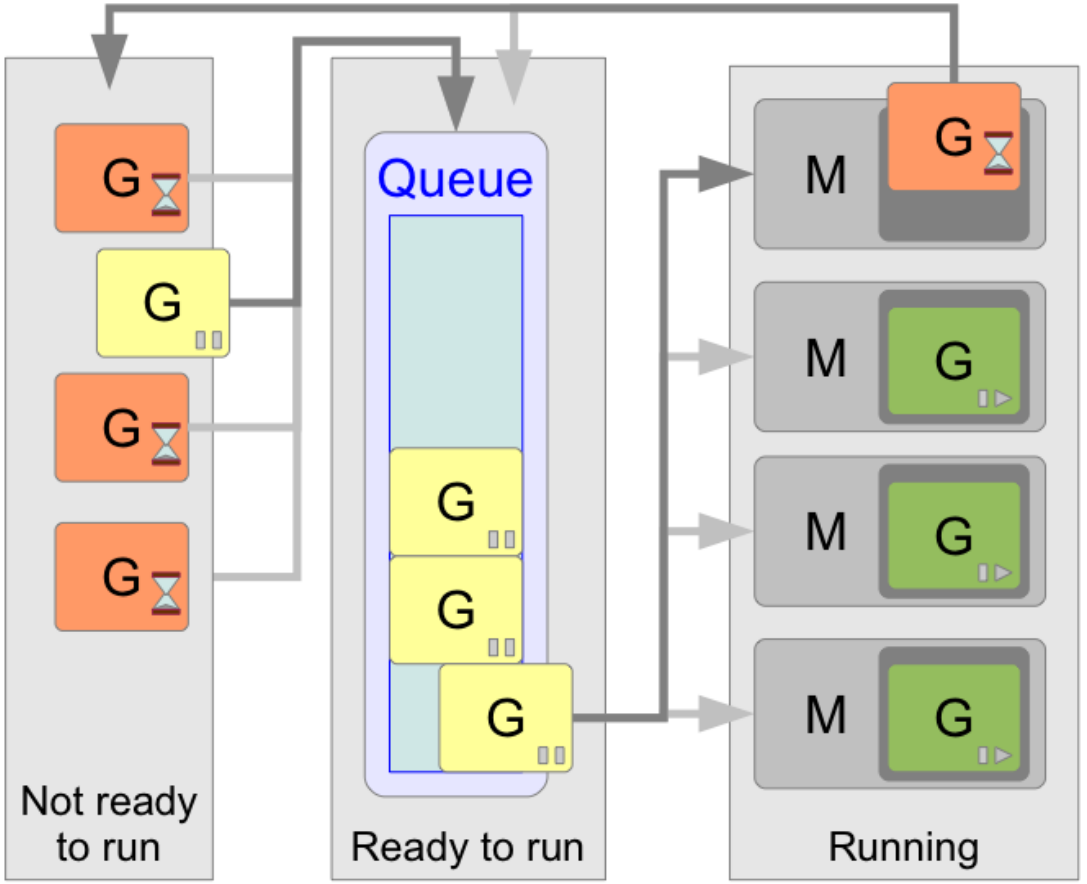
Горутины. Goroutines

Горутины:

На практике это в первую очередь означает, что иногда стоит использовать runtime.Gosched(), чтобы несколько долгоживущих горутин не остановили на существенное время работу всех других.

С другой стороны, такие ситуации встречаются на практике довольно редко.

Горутины. Goroutines



Горутины. Goroutines

Объявление горутины:

```
// анонимная горутина
go func() {

}()

// вызов функции как горутины
go myFunction()
```

Каналы. Channels

Каналы:

Основное назначение – коммуникация между горутинами с целью их синхронизации.

Исходный код каналов досутпен на github в файле chan.go, и центральной структурой данных для канала является hchan.

Каналы. Channels

```
type hchan struct {
    qcount   uint           // total data in the queue
    dataqsiz uint           // size of the circular queue
    buf      unsafe.Pointer // points to an array of dataqsiz elements
    elemsize uint16
    closed   uint32
    elemtype *_type // element type
    sendx    uint   // send index
    recvx    uint   // receive index
    recvq    waitq  // list of recv waiters
    sendq    waitq  // list of send waiters

    // lock protects all fields in hchan, as well as several
    // fields in sudogs blocked on this channel.
    //
    // Do not change another G's status while holding this lock
    // (in particular, do not ready a G), as this can deadlock
    // with stack shrinking.
    lock mutex
}
```

Каналы. Channels

```
ch := make(chan int, 2) // Создаем буферизованный канал
ch ← 42

// заполненная структура hchan
ch = {chan int}
 qcount = {uint} 1
 dataqsiz = {uint} 2
 *buf = {*[2]int} len:2
 elemsize = {uint16} 8
 closed = {uint32} 0
 *elemtype = {*runtime._type}
 sendx = {uint} 1
 recvx = {uint} 0
 recvq = {waitq<int>}
 sendq = {waitq<int>}
 lock = {runtime.mutex}
```

Каналы. Channels

- Oqcount определяет количество элементов в буфере(мы видим 1 т.к. записали одно значение в наш канал)
- dataqsiz определяет размерность буфера для буферизированного канала, в нашем случае это 2;
- buf — определяет буфер с данными, записанными в канал, реализованный с помощью структуры данных “кольцевой буфер”
- elemsize — размер одного элемента в канале, в нашем случае это 8 байт

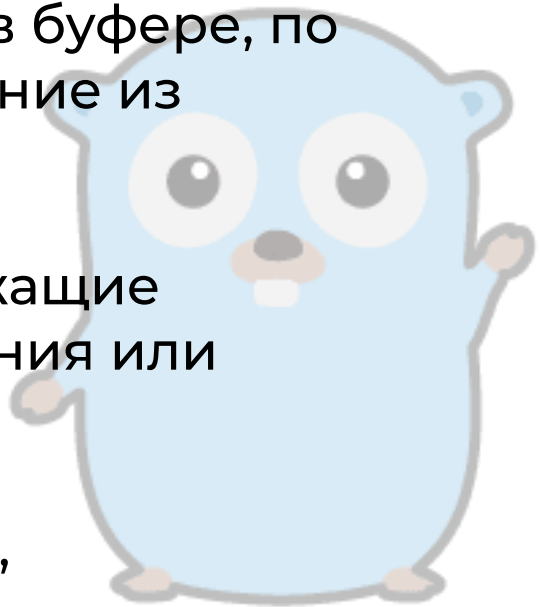
Каналы. Channels

- closed — определяет закрыт или открыт канал в данный момент;
- elemype — содержит указатель на тип данных в канале;

- sendx и recvx содержат индексы (смещения) в буфере, по которым должна производиться запись и чтение из буфера соответственно;

- sendq и recvq — односвязные списки, содержащие заблокированные горутины, ожидающие чтения или записи;

- lock — мьютекс, используемый для операций, изменяющих состояние канала



Каналы. Channels

Создание канала:

Создание канала происходит при помощи функции makechan, которая описана в том же исходном файле chan.go

Функция makechan выделяет память под структуру hchan в куче, инициализирует эту структуру, и возвращает указатель.
И несмотря на то, что в go все передается по значению, передавать канал по ссылке бессмысленно, потому что под капотом канал — это и есть указатель.