Seminario de Lenguajes opción Go

Raúl Champredonde

Seminario de Lenguajes opción Go

- Filósofos
- Barbero durmiente

Problemas de los “Dining Philosophers”

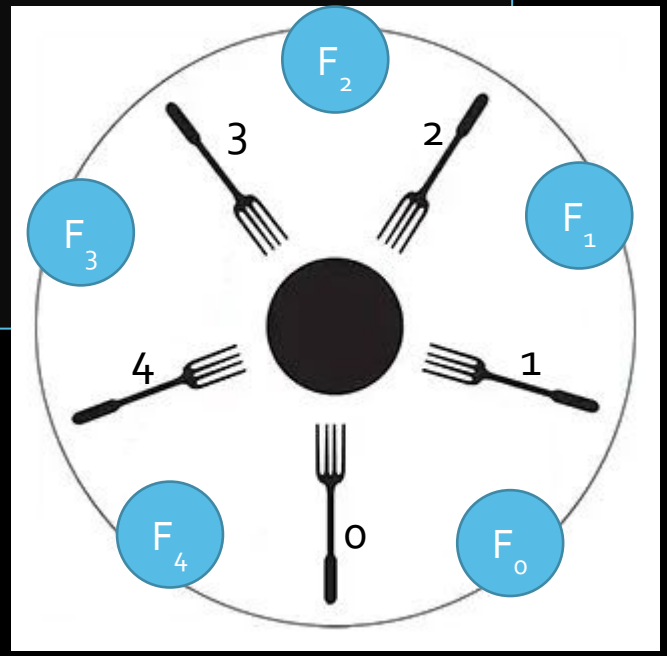
- 5 filósofos alrededor de una mesa con un bowls de spaghettis en el medio.
- 1 tenedor entre cada par de filósofos adyacentes.
- Cada filósofo alterna entre pensar y comer.
- Un filósofo solo puede comer cuando tiene ambos tenedores: el de su izquierda y el de su derecha.
- Cada tenedor solo puede ser usado por un filósofo a la vez.
- Después de comer, cada filósofo deja ambos tenedores en la mesa, quedando entonces disponibles para que los utilicen los filósofos adyacentes.
- Tanto la cantidad de spaghettis como la capacidad del estómago de los filósofos son infinitas.

Concurrencia – Dining Philosophers

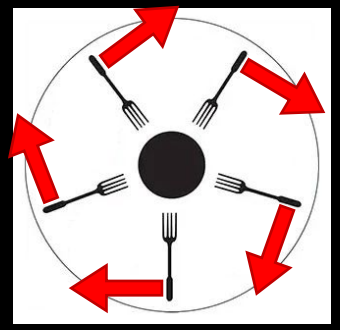
```
var philos = []string{
  "Mark",
  "Russell",
  "Rocky",
  "Haris",
  "Root",
}

var forks =
[5]sync.Mutex{}
```

```
func philosopher(id int, forkL, forkR *sync.Mutex)
{
  name := philos[id]
  // "seated"
  for i := 0; i < 50; i++ {
    // "thinking"
    forkL.Lock()
    forkR.Lock()
    // "eating"
    forkL.Unlock()
    forkR.Unlock()
  }
  // "left the table"
  dining.Done()
}
```

```
func main() {
  dining.Add(5)
  for i := range philos {
    go philosopher(i, &forks[i],
&forks[(i+1)%5])
  }
  dining.Wait()
}
```



Concurrencia - Deadlock



- Condiciones necesarias para la ocurrencia de "deadock":
  - EXCLUSIÓN MUTUA
    - Los procesos tienen acceso exclusivo a los recursos que necesitan.
  - RETENCIÓN Y ESPERA
    - Los procesos mantienen el acceso a los recursos ya asignados mientras esperan por recursos adicionales retenidos por otros procesos.
  - NO APROPIACIÓN
    - Un recurso sólo es liberado voluntariamente por el proceso que lo retiene, después que haya cumplido su tarea.
  - ESPERA CIRCULAR
    - Debe existir un conjunto de procesos (p_0,p_1,…,p_n) en espera, tales que p_0 espera un recurso retenido por p_1, p_1 espera un recurso retenido por p_2 y así sucesivamente hasta que p_n espera un recurso retenido por p_0.

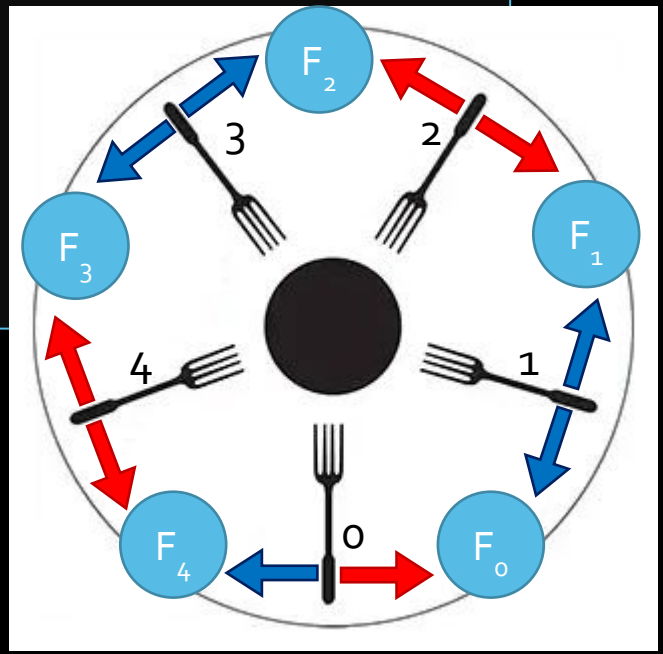
Problemas de los “Dining Philosophers”

- Evitar deadlock

```
var philos = []string{
  "Mark",
  "Russell",
  "Rocky",
  "Haris",
  "Root",
}

var forks =
[5]sync.Mutex{}
```

```
func philosopher(id int, forkL, forkR *sync.Mutex)
{
  name := philos[id]
  // "seated"
  for i := 0; i < 50; i++ {
    // "thinking"
    forkL.Lock()
    forkR.Lock()
    // "eating"
    forkL.Unlock()
    forkR.Unlock()
  }
  // "left the table"
  dining.Done()
}
```

```
func main() {
  dining.Add(5)
  for i := range philos {
    go philosopher(i, &forks[(i+i%2)%5],
&forks[(i+1-i%2)%5])
  }
  dining.Wait()
}
```



Problemas de los “Dining Philosophers”

- Inanición (starvation)

```
var philos = []string{
  "Mark",
  "Russell",
  "Rocky",
  "Haris",
  "Root",
}
var forks =
[5]sync.Mutex{}
var count = [5]int{}
var penalty = [5]int{}
```

```
func max(count ints, id int) int
{
  result := 0
  for i, v := range count {
    if i != id && v > result {
      result = v
    }
  }
  return result
}
```

```
func philosopher(id int, forkL, forkR *sync.Mutex)
{
  name := philos[id]
  // "seated"
  for i := 0; i < 50; i++ {
    // "thinking"
    time.Sleep(time.Duration(500 * penalty[id]))
    forkL.Lock()
    forkR.Lock()
    // "eating"
    forkL.Unlock()
    forkR.Unlock()
    report(id)
  }
  // "left the table"
  dining.Done()
}
```

```
func report(id int) {
  var mu sync.RWMutex
  count[id]++
  mu.RLock()
  if float64(count[id]) >
     float64(max(count, id))*1.1 {
    penalty[id]++
  } else {
    if penalty[id] > 0 {
      penalty[id]--
    }
  }
  mu.RUnlock()
}
```

Concurrencia – Barbero durmiente

- El problema del barbero:
  - El barbero tiene una silla de barbero en una sala de corte y una sala de espera que contiene varias sillas.
  - Cuando el barbero termina de cortar el cabello de un cliente, lo despide y va a la sala de espera para ver si hay más esperando.
  - Si hay alguno, llama al primero y le corta el pelo.
  - Si no hay ninguno, vuelve a la silla y duerme en ella.
  - Cada cliente, cuando llega, mira para ver qué está haciendo el barbero.
  - Si el barbero está durmiendo, el cliente lo despierta y se sienta en la silla de la sala de corte.
  - Si el barbero está atendiendo, el cliente se queda en la sala de espera.
  - Si hay una silla libre en la sala de espera, el cliente se sienta en ella y espera su turno.
  - Si no hay silla libre, el cliente se va.

Problema del “Barbero durmiente”

- Primer aproximación

```
const n = 5

func main() {
  sillas := make(chan string, n)
  despertar := make(chan string)
  listo := make(chan bool)

  clientes := []string{"Alberto", "Braian", ......}

  var wg sync.WaitGroup

  go barbero(sillas, listo, despertar)

  wg.Add(len(clientes))
  for _, nombre := range clientes {
    go func(nom string) {
      time.Sleep(time.Duration(1000 +
rand.Intn(1000)))
      cliente(nom, sillas, listo, despertar)
      wg.Done()
    }(nombre)
  }
  wg.Wait()
}
```

Problema del “Barbero durmiente”

- Primer aproximación

```
func barbero(sillas <-chan string,
             listo chan<- bool,
             despertar <-chan string) {
  for {
    // Durmiendo
    ftm.Println("- Durmiendo ...", "")
    nombre := <-despertar
    prnt(0, "- Despertado por", nombre)
    cortar(nombre, listo, sillas)
    // Me fijo si hay algún cliente esperando
    // Si hay, lo atiendo. Si no, duermo
    for len(sillas) > 0 {
      nombre := <-sillas
      prnt(6, nombre, "pasa a silla de
corte")
      cortar(nombre, listo, sillas)
    }
  }
}
```

```
func cliente(nombre string,
             sillas chan string,
             listo <-chan bool,
             despertar chan<- string) {
  prnt(6, "Llega", nombre, sillas)
  select {
  case despertar <- nombre:
    prnt(6, nombre, "despierta al barbero")
  default:
    select {
    case sillas <- nombre:
      prnt(6, nombre,
           "se sienta en sala de espera")
      <-listo
      prnt(6, nombre, "se va con el pelo corto")
    default:
      prnt(6, nombre, " no hay lugar en sala de espera")
    }
  }
}
```

```
func cortar(nombre string, listo chan<- bool, sillas <-chan string)
{
  prnt(0, "- Cortando a", nombre, sillas)
  time.Sleep(time.Duration(100 + rand.Intn(100)))
  listo <- true
  prnt(0, "- Fin del corte a", nombre, sillas)
}
```