PPADS MMXX

Matus Jokay, C-503, matus.jokay@stuba.sk
Roderik Ploszek, C-512, roderik.ploszek@stuba.sk

uim.fei.stuba.sk/predmet/i-ppds

konzultacie dohodou

- Mutex bez vyhladovenia
- Vecerajuci filozofi

LITTLE BOOK OF SEMAPHORES

- Mutex bez vyhladovenia (kapitola 4.3 No starve mutex)
- Vecerajuci filozofi (kapitola 4.4 Dining philosophers)

LITTLE BOOK OF SEMAPHORES

- Allen B. Downey: The Little Book of Semaphores, Version 2.2.1
- http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf

MUTEX BEZ VYHLADOVENIA

MUTEX BEZ VYHLADOVENIA

- Stretli sme sa s problemom vyhladovenia kategorie procesov

MUTEX BEZ VYHLADOVENIA

- Stretli sme sa s problemom vyhladovenia kategorie procesov
- Co vsak vyhladovenie vlakien tej istej kategorie?

MUTEX BEZ VYHLADOVENIA

- Stretli sme sa s problemom vyhladovenia kategorie procesov
- Co vsak vyhladovenie vlakien tej istej kategorie?
  - Mutex, semafor bez FIFO!

MUTEX BEZ VYHLADOVENIA

- Stretli sme sa s problemom vyhladovenia kategorie procesov
- Co vsak vyhladovenie vlakien tej istej kategorie?
  - Mutex, semafor bez FIFO!
- Ohranicene cakanie

MUTEX BEZ VYHLADOVENIA

- Vyhladovenie versus planovac v OS

MUTEX BEZ VYHLADOVENIA

- Vyhladovenie versus planovac v OS
  - Planovac vybera vlakno/vlakna

MUTEX BEZ VYHLADOVENIA

- Vyhladovenie versus planovac v OS
  - Planovac vybera vlakno/vlakna
  - Ak nebude FIFO, bude zle
- Predpoklady kladene na planovac

MUTEX BEZ VYHLADOVENIA

- Vlastnost 1

MUTEX BEZ VYHLADOVENIA

- Vlastnost 1
  - Ak jestvuje iba 1 vlakno, ktore moze pokracovat, vyberie ho na beh

MUTEX BEZ VYHLADOVENIA

- Vlastnost 1
  - Ak jestvuje iba 1 vlakno, ktore moze pokracovat, vyberie ho na beh
  - S tymto predpokladom si pri synchronizacii vystacime

MUTEX BEZ VYHLADOVENIA

- Vlastnost 1
  - Ak jestvuje iba 1 vlakno, ktore moze pokracovat, vyberie ho na beh
  - S tymto predpokladom si pri synchronizacii vystacime
  - Bariera…

MUTEX BEZ VYHLADOVENIA

- Vlastnost 2

MUTEX BEZ VYHLADOVENIA

- Vlastnost 2
  - Ak je vlakno pripravene na beh, potom cas, ktory musi cakat, kym pobezi, je konecny

MUTEX BEZ VYHLADOVENIA

- Vlastnost 2
  - Ak je vlakno pripravene na beh, potom cas, ktory musi cakat, kym pobezi, je konecny
  - Doteraz sme tuto vlastnost predpokladali, a aj nadalej budeme

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3
  - Ak nejake vlakno vykonava nad semaforom operaciu signal() a sucasne existuju cakajuce vlakna nad tymto semaforom, musi sa jedno z tychto cakajucich zobudit

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3
  - Ak nejake vlakno vykonava nad semaforom operaciu signal() a sucasne existuju cakajuce vlakna nad tymto semaforom, musi sa jedno z tychto cakajucich zobudit
  - Tato poziadavka sa zda zbytocna, ale…

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3
  - Ak nejake vlakno vykonava nad semaforom operaciu signal() a sucasne existuju cakajuce vlakna nad tymto semaforom, musi sa jedno z tychto cakajucich zobudit
  - Tato poziadavka sa zda zbytocna, ale predstavme si vlakno, ktore v cykle robi nad semaforom wait() – signal()

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C
  - V nekonecnom cykle

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C
  - V nekonecnom cykle

```
A / B / C
1)  while True:
2)  mutex.lock()
3)  // kod vlakna
4)  mutex.unlock()
```

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C
  - V nekonecnom cykle
  - A

A / B / C

```
1) while True:
2)     mutex.lock()
3)     // kod vlakna
4)     mutex.unlock()
```

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C
  - V nekonecnom cykle
  - A, B

```
        A / B / C
1)  while True:
2)      mutex.lock()
3)      // kod vlakna
4)      mutex.unlock()
```

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C
  - V nekonecnom cykle
  - A, B, A

```
        A / B / C
1)  while True:
2)      mutex.lock()
3)      // kod vlakna
4)      mutex.unlock()
```

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C
  - V nekonecnom cykle
  - A, B, A, B

```
        A / B / C
1)  while True:
2)      mutex.lock()
3)      // kod vlakna
4)      mutex.unlock()
```

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C
  - V nekonecnom cykle
  - A, B, A, B, …

```
        A / B / C
1)  while True:
2)      mutex.lock()
3)      // kod vlakna
4)      mutex.unlock()
```

MUTEX BEZ VYHLADOVENIA

- Vlastnost 3 samotna nezarucuje to, ze nenastane vyhladovenie
  - Majme 3 vlakna: A, B, C
  - V nekonecnom cykle
  - A, B, A, B, …
  - C stale caka ;)

```
        A / B / C
1)  while True:
2)      mutex.lock()
3)      // kod vlakna
4)      mutex.unlock()
```

MUTEX BEZ VYHLADOVENIA

- Vlastnost 4

MUTEX BEZ VYHLADOVENIA

- Vlastnost 4
  - Ak caka vlakno na semafore/zamku, pocet vlakien, ktore sa zobudia pred nim, je konecny

MUTEX BEZ VYHLADOVENIA

- Vlastnost 4
  - Ak caka vlakno na semafore/zamku, pocet vlakien, ktore sa zobudia pred nim, je konecny
  - Napr. FIFO fronta cakajucich

MUTEX BEZ VYHLADOVENIA

- Semafor s vlastnostou 3 sa nazyva “slaby” semafor
- Semafor s vlastnostou 4 sa nazyva “silny” semafor

MUTEX BEZ VYHLADOVENIA

- Jestvuje moznost implementovat mutex bez vyhladovenia iba s vlastnostou 3?

MUTEX BEZ VYHLADOVENIA

- Jestvuje moznost implementovat mutex bez vyhladovenia iba s vlastnostou 3?
- Ujo Dijkstra povedal, ze to nie je mozne

MUTEX BEZ VYHLADOVENIA

- Jestvuje moznost implementovat mutex bez vyhladovenia iba s vlastnostou 3?
- Ujo Dijkstra povedal, ze to nie je mozne
- Ale ujo Morris ukazal, ze to mozne je

MUTEX BEZ VYHLADOVENIA

- Jestvuje moznost implementovat mutex bez vyhladovenia iba s vlastnostou 3?
- Ujo Dijkstra povedal, ze to nie je mozne
- Ale ujo Morris ukazal, ze to mozne je
  - Ak je pocet vlakien vopred dany a konecny

MUTEX BEZ VYHLADOVENIA

- Myslienka ako pri znovupouzitelnej bariere

MUTEX BEZ VYHLADOVENIA

- Myslienka ako pri znovupouzitelnej bariere
- 2 turnikety (dvere), ktore oddeluju 2 miestnosti pred vstupom do KO

MUTEX BEZ VYHLADOVENIA

- Myslienka ako pri znovupouzitelnej bariere
- 2 turnikety (dvere), ktore oddeluju 2 miestnosti pred vstupom do KO
- Mutex funguje na 2 fazy

MUTEX BEZ VYHLADOVENIA

- 1. faza

MUTEX BEZ VYHLADOVENIA

- 1. faza
  - Prve dvere (z miestnosti 1 do 2) su otvorene

MUTEX BEZ VYHLADOVENIA

- 1. faza
  - Prve dvere (z miestnosti 1 do 2) su otvorene
  - Druhe dvere (z miestnosti 2 do KO) su zatvorene

MUTEX BEZ VYHLADOVENIA

- 1. faza
  - Prve dvere (z miestnosti 1 do 2) su otvorene
  - Druhe dvere (z miestnosti 2 do KO) su zatvorene
  - Co znamena, ze vsetky vlakna sa nutne musia zhromazdit v miestnosti c. 2

MUTEX BEZ VYHLADOVENIA

- 2. faza

MUTEX BEZ VYHLADOVENIA

- 2. faza
  - Prve dvere (z miestnosti 1 do 2) su zatvorene

MUTEX BEZ VYHLADOVENIA

- 2. faza
  - Prve dvere (z miestnosti 1 do 2) su zatvorene
  - Druhe dvere (z miestnosti 2 do KO) su otvorene

MUTEX BEZ VYHLADOVENIA

- 2. faza
  - Prve dvere (z miestnosti 1 do 2) su zatvorene
  - Druhe dvere (z miestnosti 2 do KO) su otvorene
  - Co znamena, ze vsetky vlakna musia vyprazdnit miestnost 2

MUTEX BEZ VYHLADOVENIA

Init()

1) room1 = room2 = 0
2) mutex = Mutex()
3) turn1 = Sem(1)
4) turn2 = Sem(0)

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
16
17
18
19
20
21
22
23
24
25
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
16
17
18
19
20      # critical section
21
22
23
24
25
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
16
17
18
19
20      # critical section
21
22
23
24
25
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6
7
8
9
10
11
12
13
14
15
```

```
16
17
18
19
20      # critical section
21
22
23
24
25
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6
7
8
9
10
11
12
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Pred KO mame 2 miestnosti

```
1
2       room1 += 1
3
4
5   turn1.wait()
6
7
8
9
10
11
12
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6
7
8
9
10
11
12
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Pred KO mame 2 miestnosti

Pri vstupe do prvej zvysujeme pocitadlo

MUTEX BEZ VYHLADOVENIA

```
1
2       room1 += 1
3
4
5   turn1.wait()
6
7
8
9
10
11
12
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Pred KO mame 2 miestnosti

Pri vstupe do prvej zvysujeme pocitadlo

Kvoli turn2 sa nemozeme ovsem dostat do KO

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6
7
8       room1 -= 1
9
10
11
12
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Pri vstupe do miestnosti 2 znizujeme pocitadlo vlakien v miestnosti 1

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10
11
12
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Pri vstupe do miestnosti 2 znizujeme pocitadlo vlakien v miestnosti 1

Zaroven zvysujeme pocitadlo vlakien pritomnych v miestnosti 2

MUTEX BEZ VYHLADOVENIA

```
1
2       room1 += 1
3
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10
11
12
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Zvysujeme pocitadlo vlakien pritomnych v miestnosti 2

Vsimnime si, ze vzhladom na hodnotu turn1 zvysovanie room2 robi vzdy iba 1 vlakno!!!

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Ak uz vsetky vlakna presli z miestnosti 1 do miestnosti 2

```
1
2       room1 += 1
3
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10      if room1 == 0
11
12
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Ak uz vsetky vlakna presli z miestnosti 1 do miestnosti 2

Otvorime dvere z miestnosti 2 do KO!

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else
14
15
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Ak vsak este nepresli…

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else:
14
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Ak vsak este nepresli…

Pustime cez dvere dalsie vlakno; vlakna chodia cez dvere zasadne po jednom! (princip turniket)

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else:
14
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Pocitadlo room1 vsak mozu spristupnovat viacere vlakna naraz!

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1
2       room1 += 1
3
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else:
14
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Pocitadlo room1 vsak mozu spristupnovat viacere vlakna naraz!

Preto pridame mutex, ktory bude chranit integritu jeho hodnoty

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else:
14
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Pridame mutex, ktory bude chranit integritu hodnoty room1

MUTEX BEZ VYHLADOVENIA

```
1    mutex.lock()
2        room1 += 1
3    mutex.unlock()
4
5    turn1.wait()
6        room2 += 1
7
8        room1 -= 1
9
10       if room1 == 0:
11
12           turn2.signal()
13       else:
14
15           turn1.signal()
```

```
16
17   turn2.wait()
18
19
20       # critical section
21
22
23
24
25
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Vieme, ze kedze vlakna chodia do miestnosti po jednom, tak iba jedno vykonava kod za riadkom 5

MUTEX BEZ VYHLADOVENIA

```
1    mutex.lock()
2        room1 += 1
3    mutex.unlock()
4
5    turn1.wait()
6        room2 += 1
7
8        room1 -= 1
9
10       if room1 == 0:
11
12           turn2.signal()
13       else:
14
15           turn1.signal()
```

```
16
17   turn2.wait()
18
19
20       # critical section
21
22
23
24
25
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Vieme, ze kedze vlakna chodia do miestnosti po jednom, tak iba jedno vykonava kod za riadkom 5

Preto netreba riesit integritu pocitadla room2

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else:
14
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Ale toto jedno vlakno meni hodnotu room1, ku ktorej pristupuju ine vlakna prichadzajuce do miestnosti 1!

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else:
14
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Ale toto jedno vlakno meni hodnotu room1, ku ktorej pristupuju ine vlakna prichadzajuce do miestnosti 1!

Preto pridame ochranu integrity room1…

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else:
14
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Nestaci mat ochranu iba pri modifikacii, ale aj pri testovani

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11
12          turn2.signal()
13      else:
14
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Nestaci mat ochranu iba pri modifikacii, ale aj pri testovani

Takze mutex ostava zamknuty aj pre riadok 10

MUTEX BEZ VYHLADOVENIA

```
1    mutex.lock()
2        room1 += 1
3    mutex.unlock()
4
5    turn1.wait()
6        room2 += 1
7        mutex.lock()
8        room1 -= 1
9
10       if room1 == 0:
11           mutex.unlock()
12           turn2.signal()
13       else:
14           mutex.unlock()
15           turn1.signal()
```

```
16
17   turn2.wait()
18
19
20       # critical section
21
22
23
24
25
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Nestaci mat ochranu iba pri modifikacii, ale aj pri testovani

Takze mutex ostava zamknuty aj pre riadok 10

Avsak po teste hodnoty ho uz mozeme odomknut

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18
19
20      # critical section
21
22
23
24
25
```

Vlakno, ktore uspesne prejde riadkom 17, je vzdy len !!!jedno!!!

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22
23
24
25
```

Vlakno, ktore uspesne prejde riadkom 17, je vzdy len !!!jedno!!!

Takze mozeme bezpecne urobit tuto operaciu?

MUTEX BEZ VYHLADOVENIA

```
1    mutex.lock()
2        room1 += 1
3    mutex.unlock()
4
5    turn1.wait()
6        room2 += 1
7        mutex.lock()
8        room1 -= 1
9
10       if room1 == 0:
11           mutex.unlock()
12           turn2.signal()
13       else:
14           mutex.unlock()
15           turn1.signal()
```

```
16
17   turn2.wait()
18       room2 -= 1
19
20       # critical section
21
22
23
24
25
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Vlakno, ktore uspesne prejde riadkom 17, je vzdy len !!!jedno!!!

Takze mozeme bezpecne urobit tuto operaciu?

Co vlakno, ktore by vykonavalo riadok 6?!

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22
23
24
25
```

Vlakno, ktore preslo z miestnosti 2, znizilo pocet vlakien v miestnosti 2

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22
23
24
25
```

Vlakno, ktore preslo z miestnosti 2, znizilo pocet vlakien v miestnosti 2

Moze ako jedine vykonat KO

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22
23
24
25
```

Vlakno, ktore preslo z miestnosti 2, znizilo pocet vlakien v miestnosti 2

Moze ako jedine vykonat KO

A potom sa pridat ku vlaknam, ktore cakaju v miestnosti 1

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22
23
24
25
```

Kedze hodnota turn1 je 0, tak vsetky vlakna po vykonani KO budu cakat na turn1

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22
23
24
25
```

Kedze hodnota turn1 je 0, tak vsetky vlakna po vykonani KO budu cakat na turn1

Aby sme sa vyhli uviaznutiu, posledne vlakno, ktore vykona KO, musi otvorit dvere z miestnosti 1 do miestnosti 2

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0
23
24
25
```

Kedze hodnota turn1 je 0, tak vsetky vlakna po vykonani KO budu cakat na turn1

Aby sme sa vyhli uviaznutiu, posledne vlakno, ktore vykona KO, musi otvorit dvere z miestnosti 1 do miestnosti 2

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24
25
```

Kedze hodnota turn1 je 0, tak vsetky vlakna po vykonani KO budu cakat na turn1

Aby sme sa vyhli uviaznutiu, posledne vlakno, ktore vykona KO, musi otvorit dvere z miestnosti 1 do miestnosti 2

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24      else
25
```

Ale ked toto vlakno, ktore prave dokoncilo KO nebolo este posledne…

MUTEX BEZ VYHLADOVENIA

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24      else:
25          turn2.signal()
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Ale ked toto vlakno, ktore prave dokoncilo KO nebolo este posledne…

Musi vpustit dalsie z cakajucich vlakien v miestnosti 2 do KO

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24      else:
25          turn2.signal()
```

Kazde vlakno, ktore dokoncilo KO, caka na riadku 5

MUTEX BEZ VYHLADOVENIA

```
1    mutex.lock()
2        room1 += 1
3    mutex.unlock()
4
5    turn1.wait()
6        room2 += 1
7        mutex.lock()
8        room1 -= 1
9
10       if room1 == 0:
11           mutex.unlock()
12           turn2.signal()
13       else:
14           mutex.unlock()
15           turn1.signal()
```

```
16
17   turn2.wait()
18       room2 -= 1
19
20       # critical section
21
22       if room2 == 0:
23           turn1.signal()
24       else:
25           turn2.signal()
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Kazde vlakno, ktore dokoncilo KO, caka na riadku 5

Kedze vlakna chodia cez dvere po jednom (vdaka turniketom), postupne odbuda pocet vlakien v miestnosti 2

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Je jedno, v akom poradi z miestnosti 2 idu…

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24      else:
25          turn2.signal()
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24      else:
25          turn2.signal()
```

Je jedno, v akom poradi z miestnosti 2 idu…

Algoritmus zarucuje, ze pred kazdym vlaknom ide vzdy konecny pocet vlakien predtym, nez samo vstupi do KO

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24      else:
25          turn2.signal()
```

Pri konecnom pocte vlakien by vyhladovenie bolo mozne, iba keby nejake dokazalo “obehnut” ostatne

MUTEX BEZ VYHLADOVENIA

```
1    mutex.lock()
2        room1 += 1
3    mutex.unlock()
4
5    turn1.wait()
6        room2 += 1
7        mutex.lock()
8        room1 -= 1
9
10       if room1 == 0:
11           mutex.unlock()
12           turn2.signal()
13       else:
14           mutex.unlock()
15           turn1.signal()
```

```
16
17   turn2.wait()
18       room2 -= 1
19
20       # critical section
21
22       if room2 == 0:
23           turn1.signal()
24       else:
25           turn2.signal()
```

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

Pri konecnom pocte vlakien by vyhladovenie bolo mozne, iba keby nejake dokazalo “obehnut” ostatne

Co pri dvojitej bariere nie je mozne

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24      else:
25          turn2.signal()
```

Pri konecnom pocte vlakien by vyhladovenie bolo mozne, iba keby nejake dokazalo “obehnut” ostatne

Co pri dvojitej bariere nie je mozne

Takze tento algoritmus riesi problem mutexu bez vyhladovenia

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
```

Implementacia silneho mutexu pomocou slabych mutexov / semaforov

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1    mutex.lock()
2        room1 += 1
3    mutex.unlock()
4
5    turn1.wait()
6        room2 += 1
7        mutex.lock()
8        room1 -= 1
9
10       if room1 == 0:
11           mutex.unlock()
12           turn2.signal()
13       else:
14           mutex.unlock()
15           turn1.signal()
```

```
16
17   turn2.wait()
18       room2 -= 1
19
20       # critical section
21
22       if room2 == 0:
23           turn1.signal()
24       else:
25           turn2.signal()
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11
12
13
14
15
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          turn2.signal()
12
13
14
15
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          turn2.signal()
12      else:
13
14
15
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          turn2.signal()
12      else:
13          turn1.signal()
14
15
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          turn2.signal()
12      else:
13          turn1.signal()
14      mutex.unlock()
15
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          turn2.signal()
12      else:
13          turn1.signal()
14      mutex.unlock()
15
```

MUTEX BEZ VYHLADOVENIA

```
Init()
room1 = room2 = 0
mutex = Mutex()
turn1 = Sem(1)
turn2 = Sem(0)
```

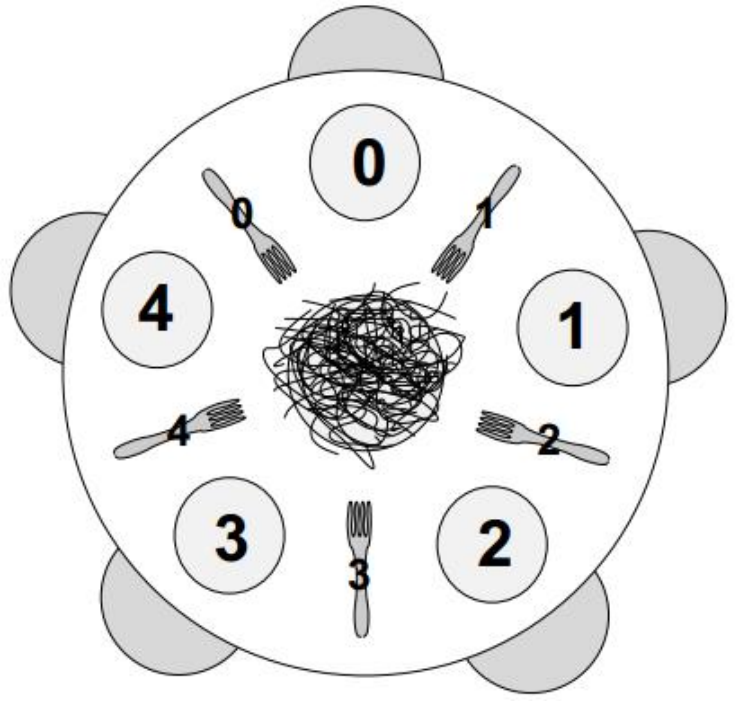
```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          mutex.unlock()
12          turn2.signal()
13      else:
14          mutex.unlock()
15          turn1.signal()
```

```
1   mutex.lock()
2       room1 += 1
3   mutex.unlock()
4
5   turn1.wait()
6       room2 += 1
7       mutex.lock()
8       room1 -= 1
9
10      if room1 == 0:
11          turn2.signal()
12      else:
13          turn1.signal()
14      mutex.unlock()
15
```

```
16
17  turn2.wait()
18      room2 -= 1
19
20      # critical section
21
22      if room2 == 0:
23          turn1.signal()
24      else:
25          turn2.signal()
```

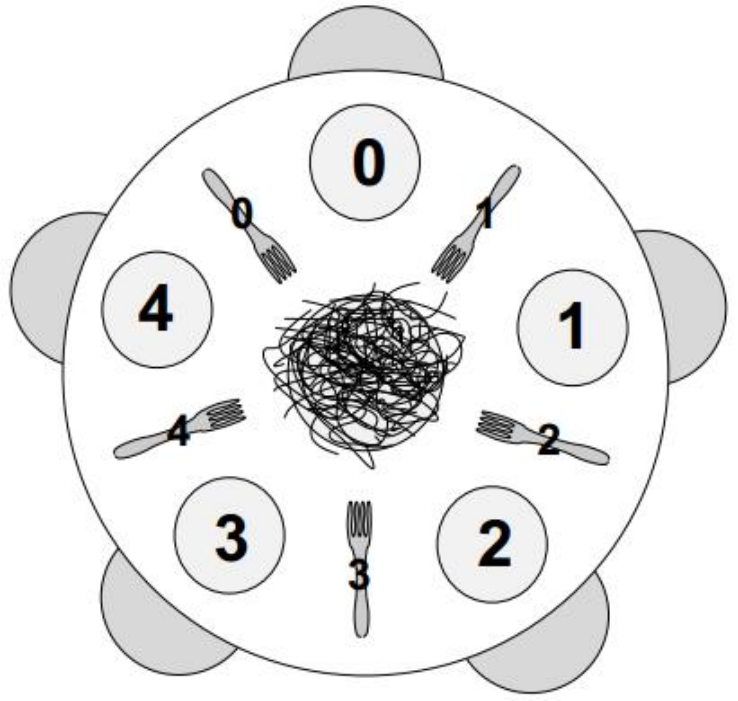
VECERAJUCI FILOZOFI

VECERAJUCI FILOZOFI



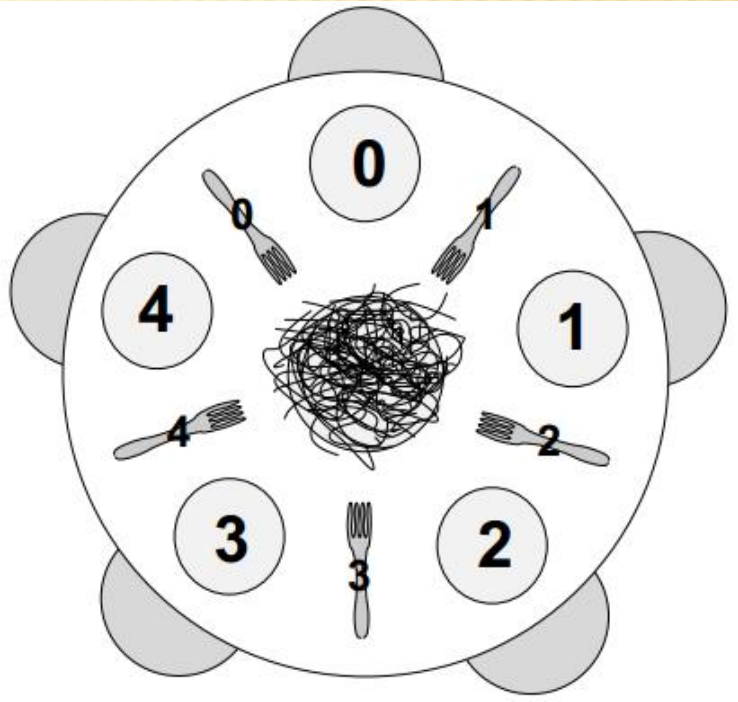
VECERAJUCI FILOZOFI

```
1    while True
```



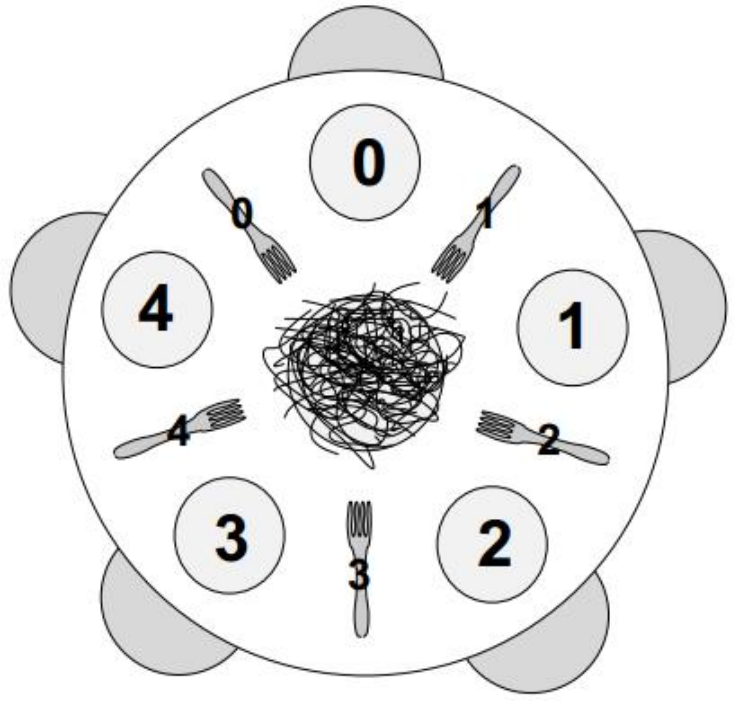
VECERAJUCI FILOZOFI

```
1    while True:
2        think()
```



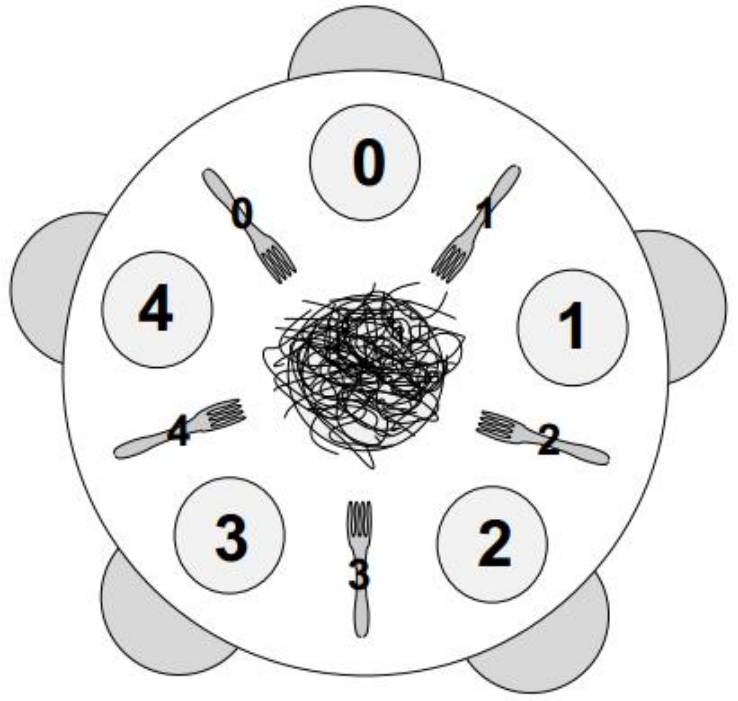
VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
```



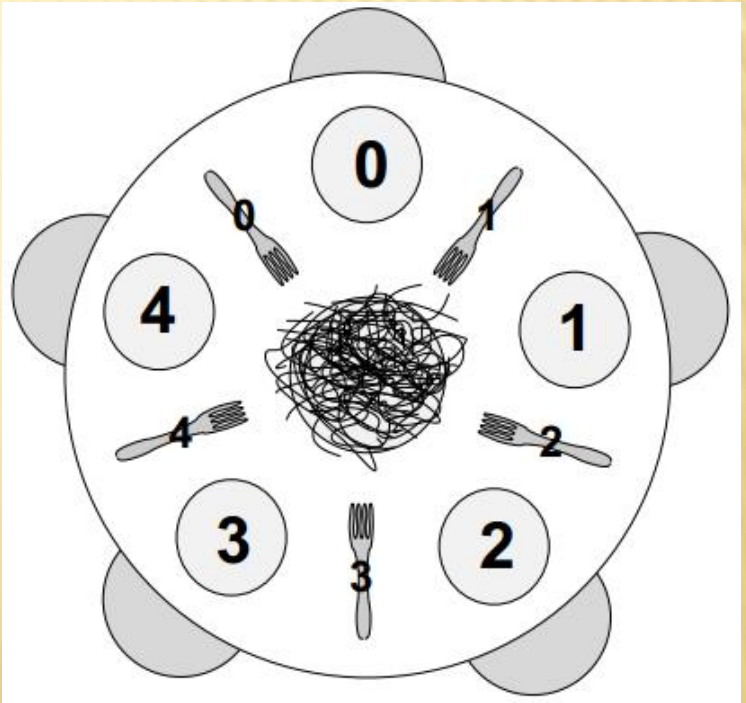
VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
```



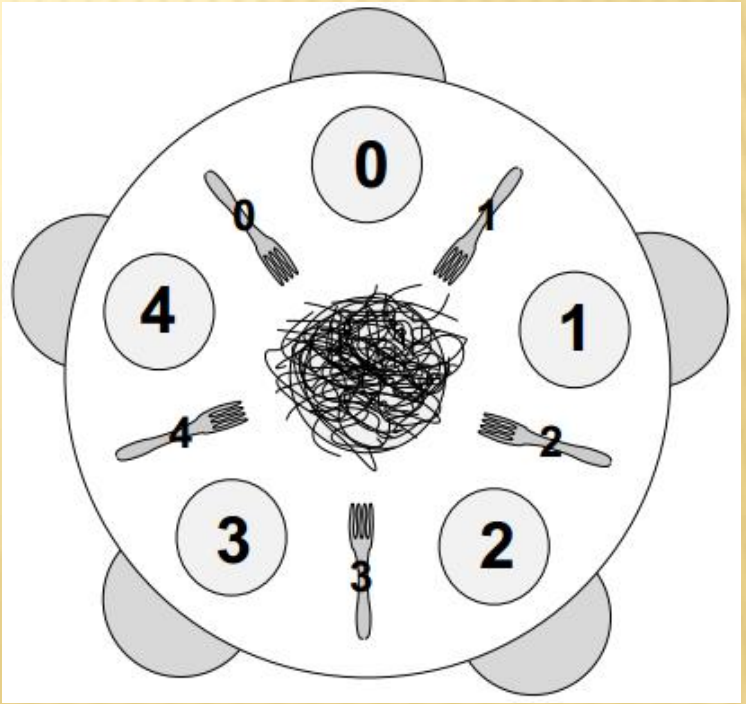
VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```



VECERAJUCI FILOZOFI

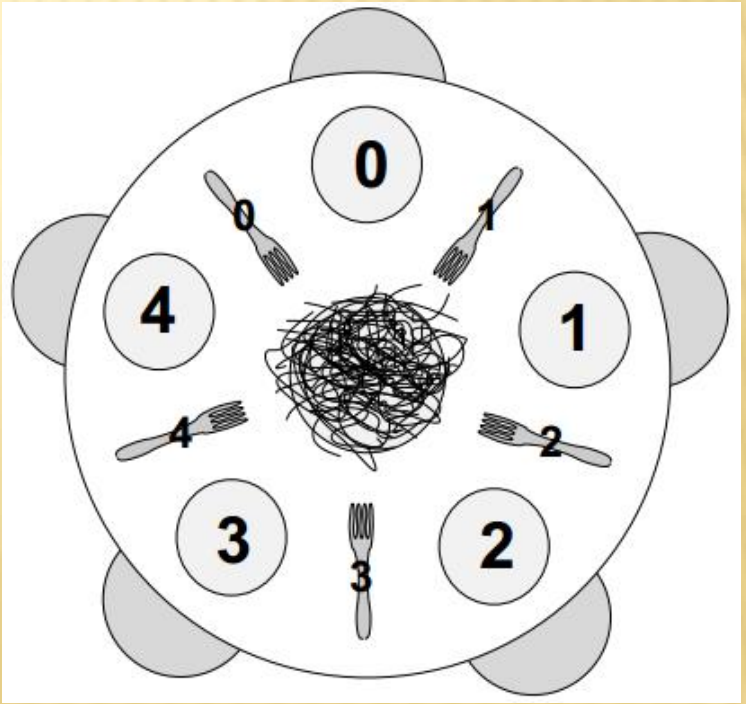
```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```



VECERAJUCI FILOZOFI

- Filozof → vlakno

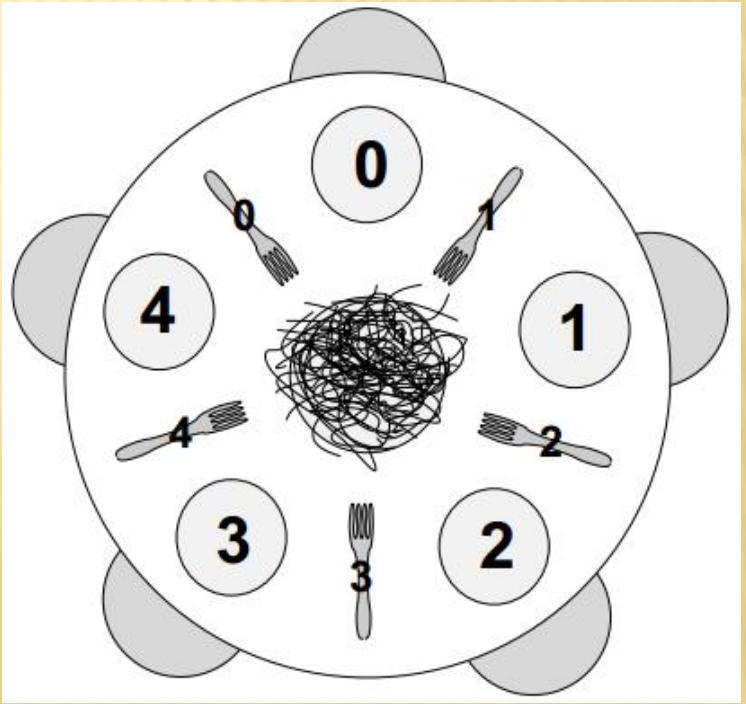
```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```



VECERAJUCI FILOZOFI

- Filozof → vlakno
- Vidlicka → zdroj

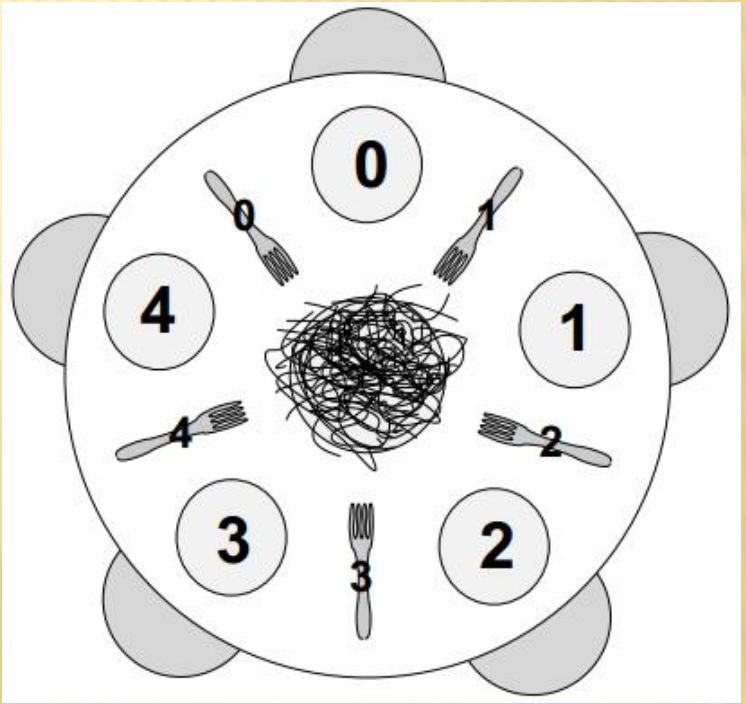
```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```



VECERAJUCI FILOZOFI

- Filozof → vlakno
- Vidlicka → zdroj
- Co je divne

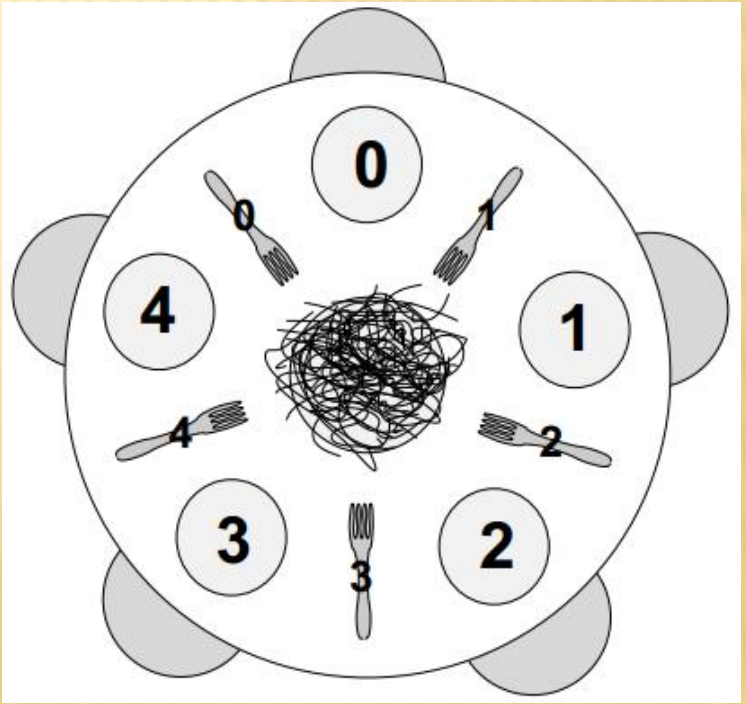
```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

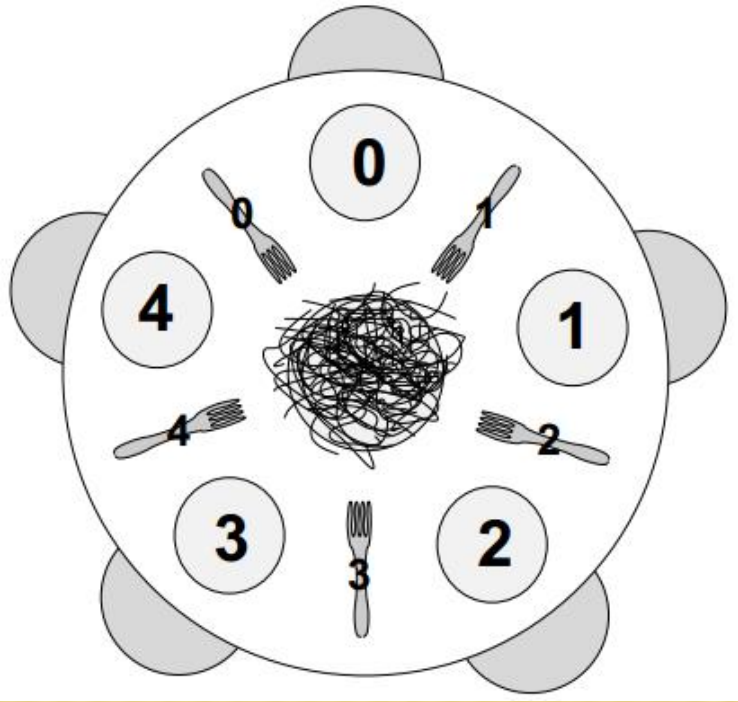
- Filozof → vlakno
- Vidlicka → zdroj
- Co je divne
  - Filozof potrebuje 2 vidlicky



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

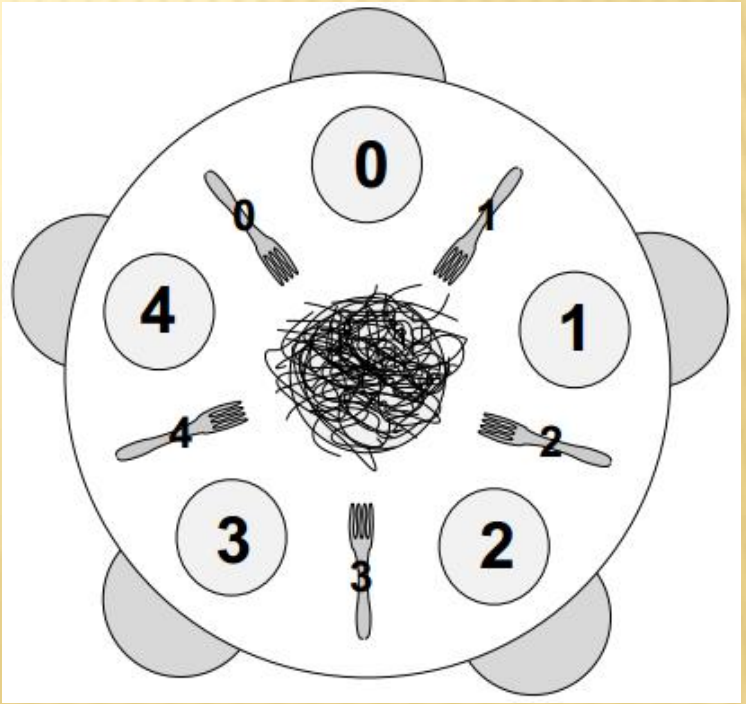
- Filozof → vlakno
- Vidlicka → zdroj
- Co je divne
  - Filozof potrebuje 2 vidlicky
  - Ak ma jednu, musi cakat na druhu, aby mohol pokracovat



VECERAJUCI FILOZOFI

- Filozof i

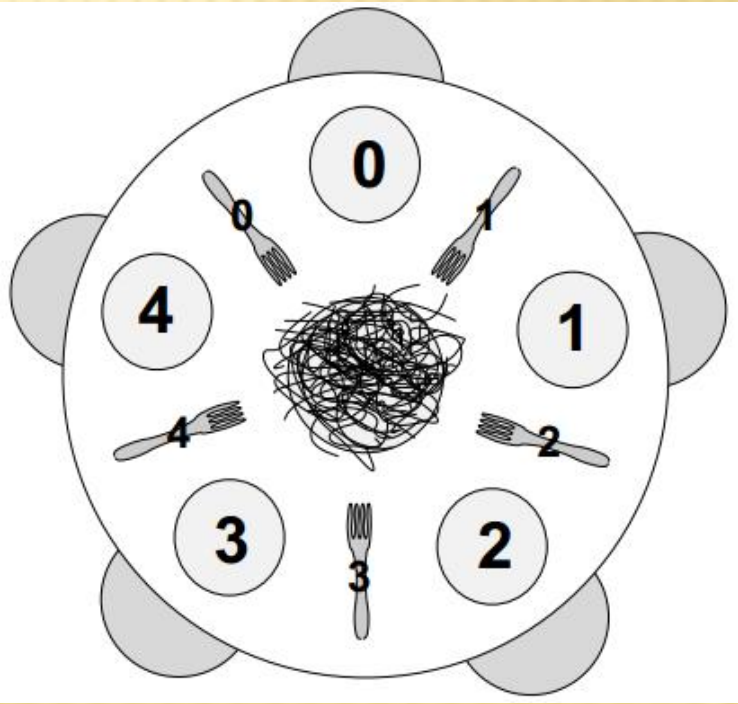
```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

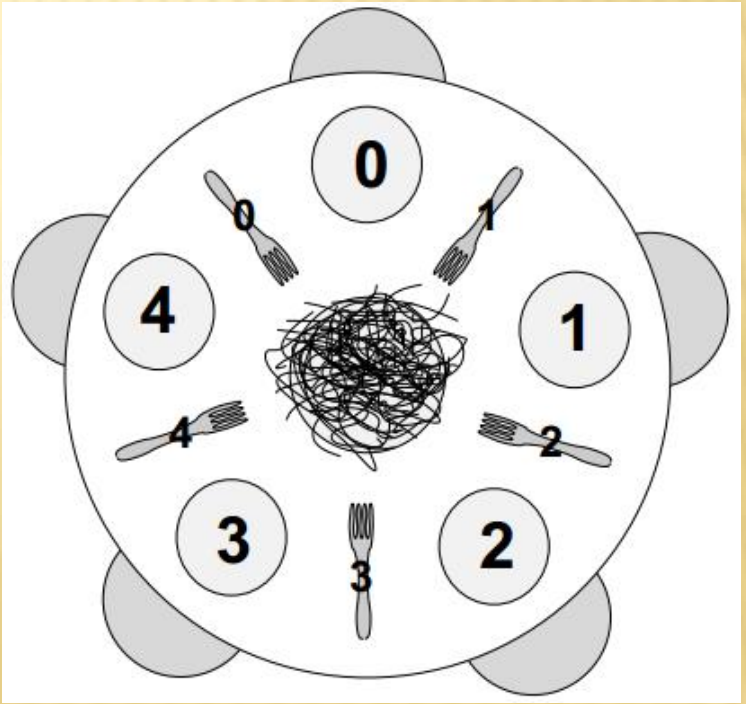
- Filozof i
  - Po pravej strane vidlicka i



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

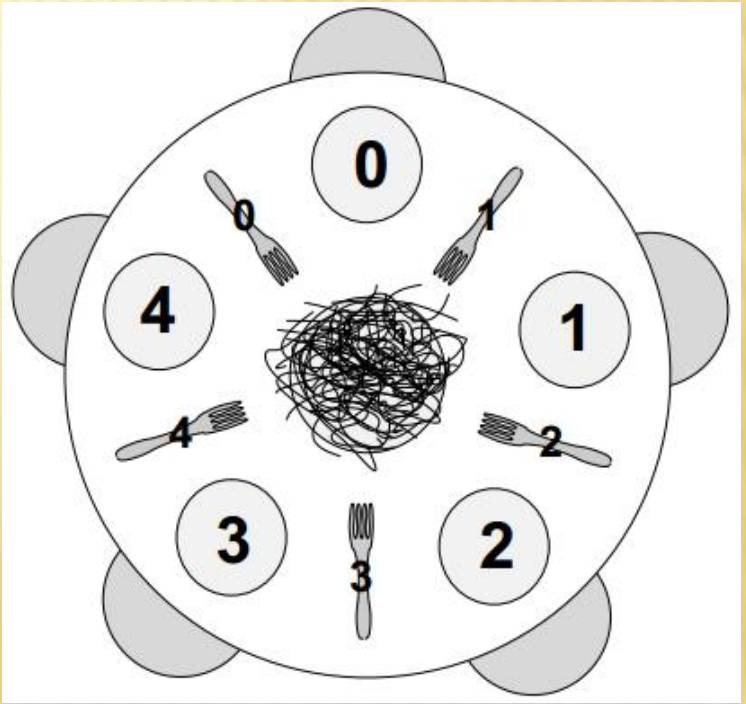
- Filozof i
  - Po pravej strane vidlicka i
  - Po lavej strane vidlicka i+1



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

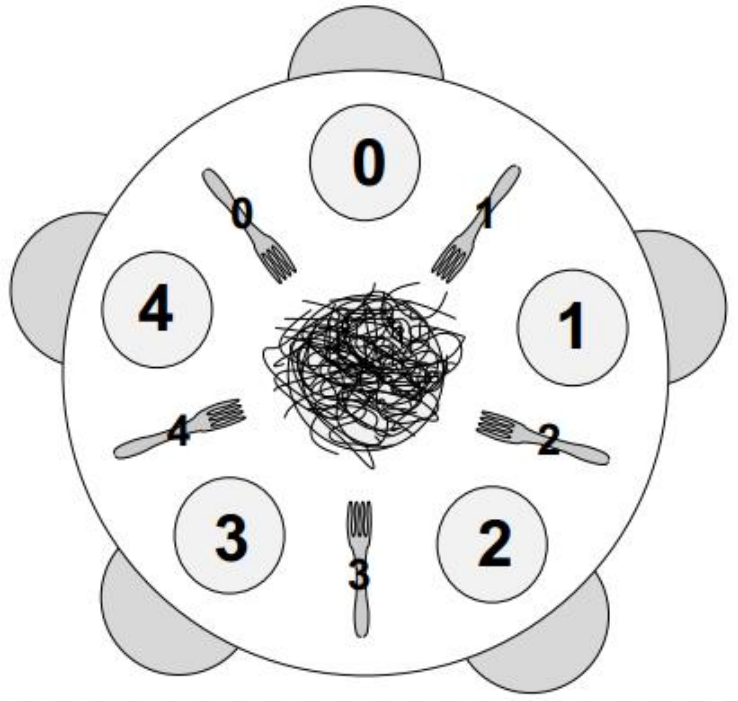
- Filozof i
  - Po pravej strane vidlicka i
  - Po lavej strane vidlicka i+1
- i v intervale <0; 4>



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

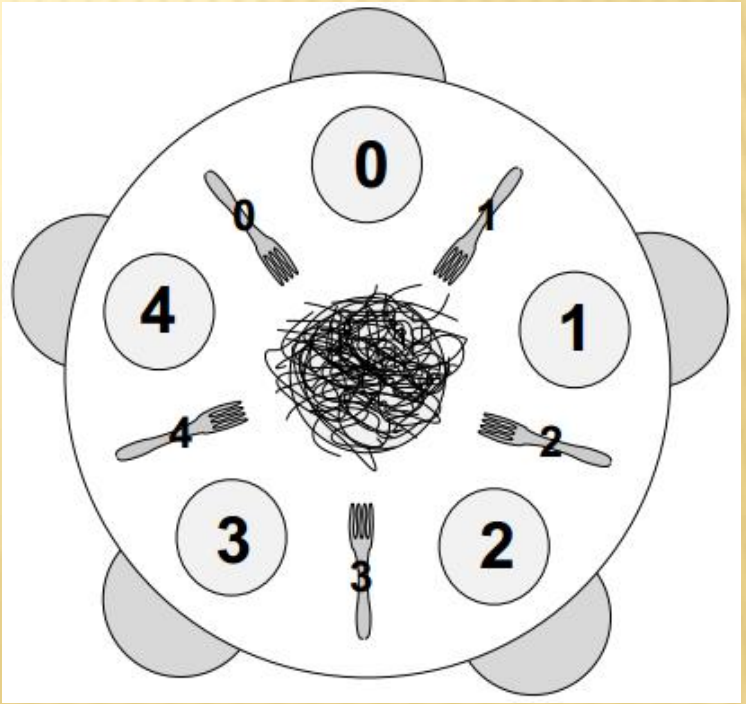
- Filozof i
  - Po pravej strane vidlicka i
  - Po lavej strane vidlicka i+1
- i v intervale <0; 4>
- Filozof ovlada funkcie
  - think()
  - eat()



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

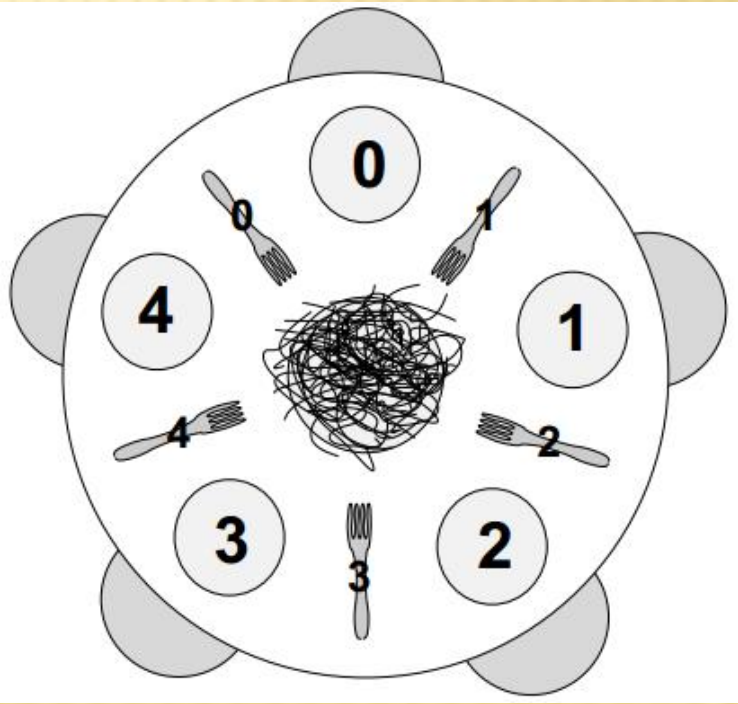
- Nasa uloha definovat get() a put() tak, aby:



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

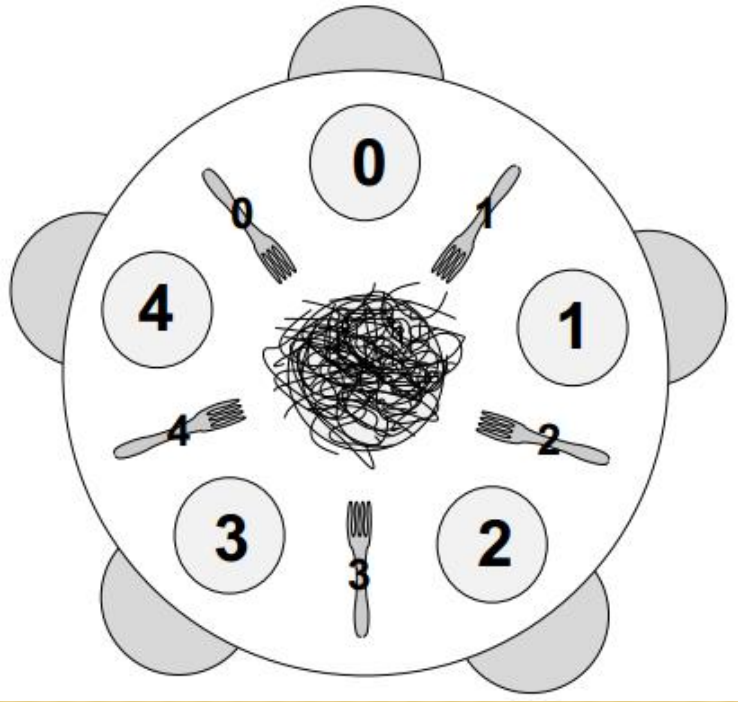
- Nasa uloha definovat get() a put() tak, aby:
  1. V 1 case drzal vidlicku iba jeden filozof



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

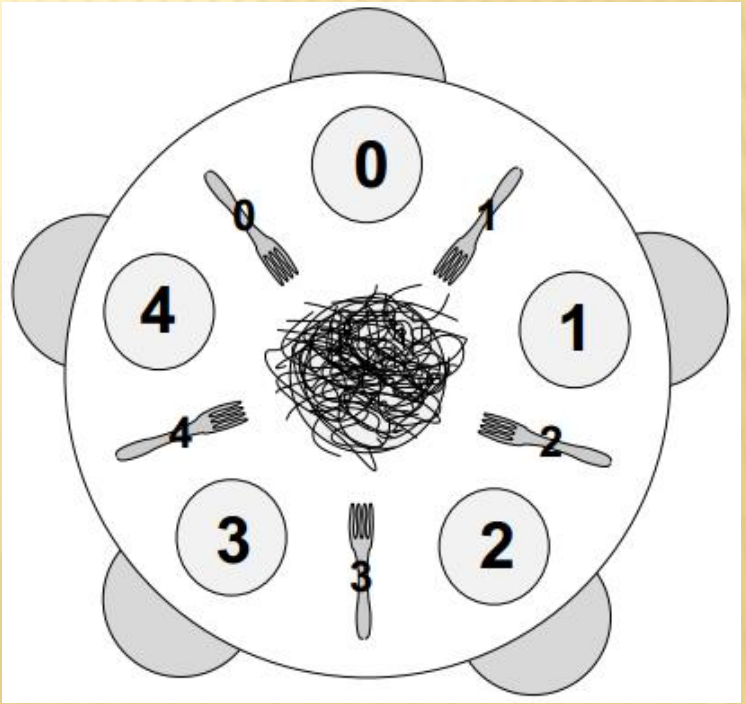
- Nasa uloha definovat get() a put() tak, aby:
  1. V 1 case drzal vidlicku iba jeden filozof
  2. Nenastalo uviaznutie



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

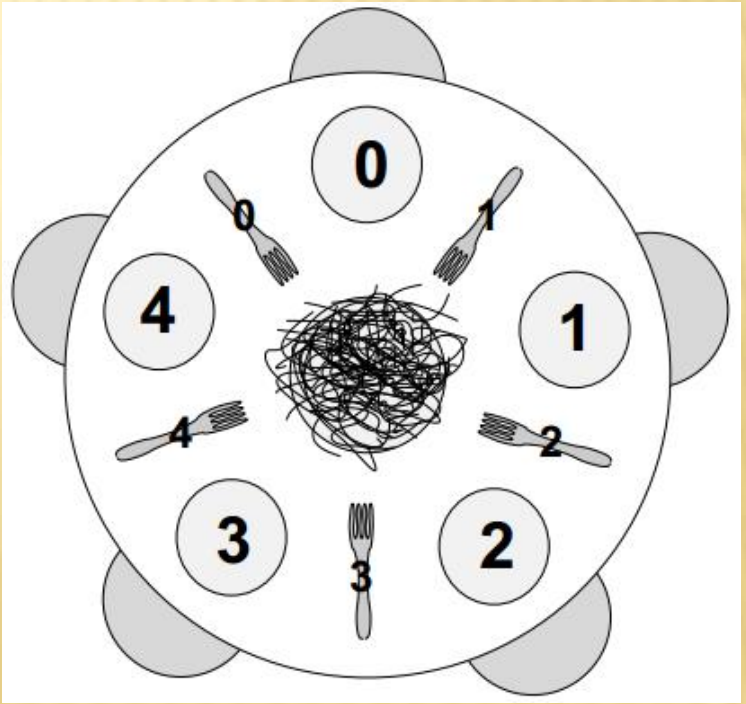
- Nasa uloha definovat get() a put() tak, aby:
  1. V 1 case drzal vidlicku iba jeden filozof
  2. Nenastalo uviaznutie
  3. Filozof neumrel od hladu



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

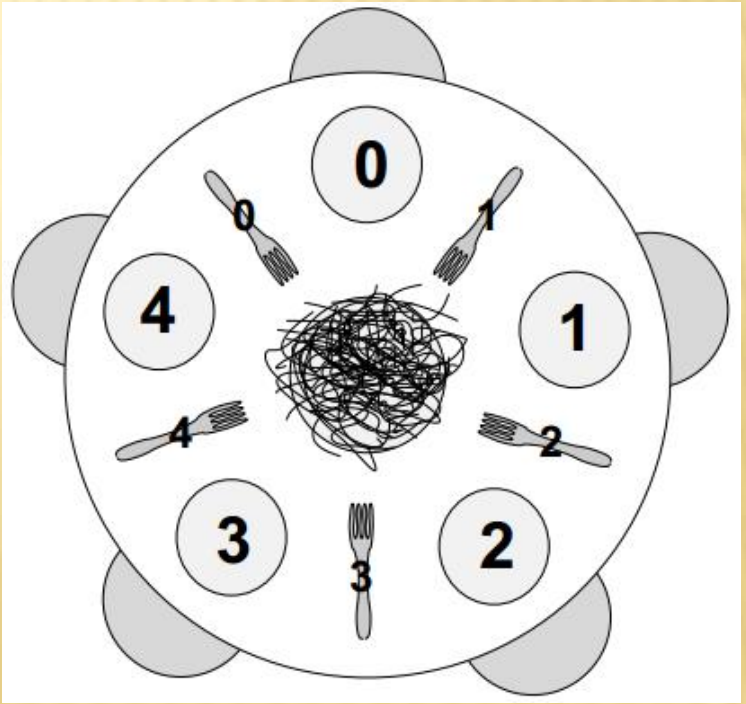
- Nasa uloha definovat get() a put() tak, aby:
  1. V 1 case drzal vidlicku iba jeden filozof
  2. Nenastalo uviaznutie
  3. Filozof neumrel od hladu
  4. Mohli jest viaceri filozofi sucasne



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

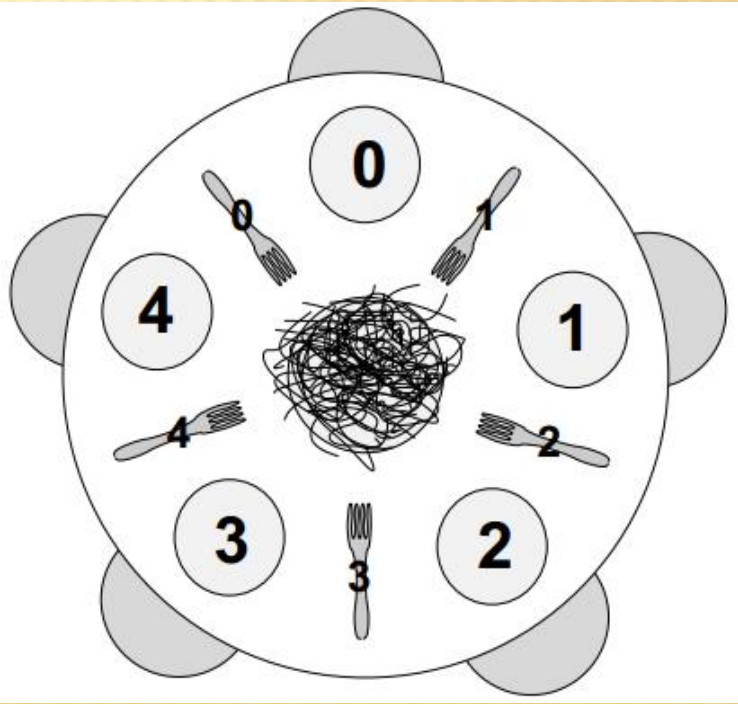
- Nasa uloha definovat get() a put() tak, aby:
  1. V 1 case drzal vidlicku iba jeden filozof
  2. Nenastalo uviaznutie
  3. Filozof neumrel od hladu
  4. Mohli jest viaceri filozofi sucasne (konkurentne)



VECERAJUCI FILOZOFI

- Nepozname funkcie think() a eat()

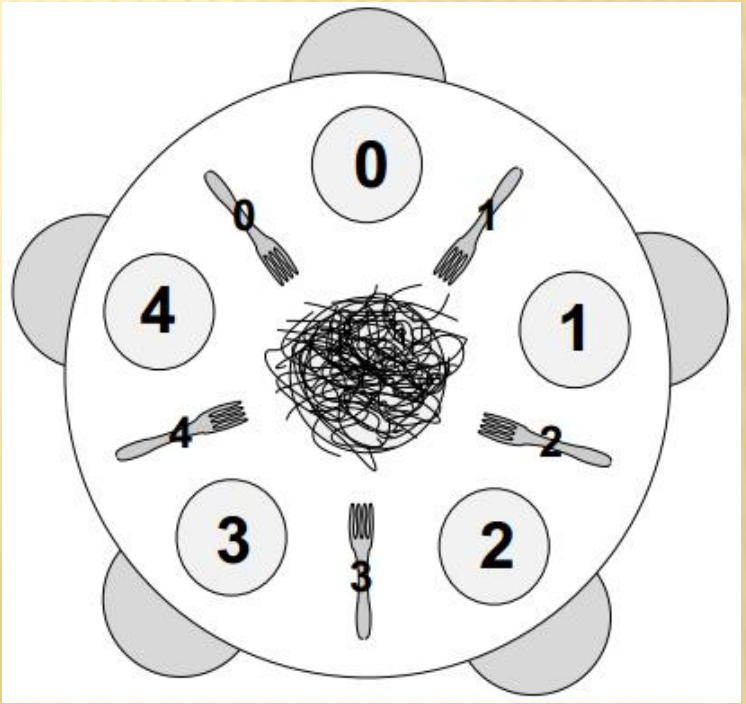
```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

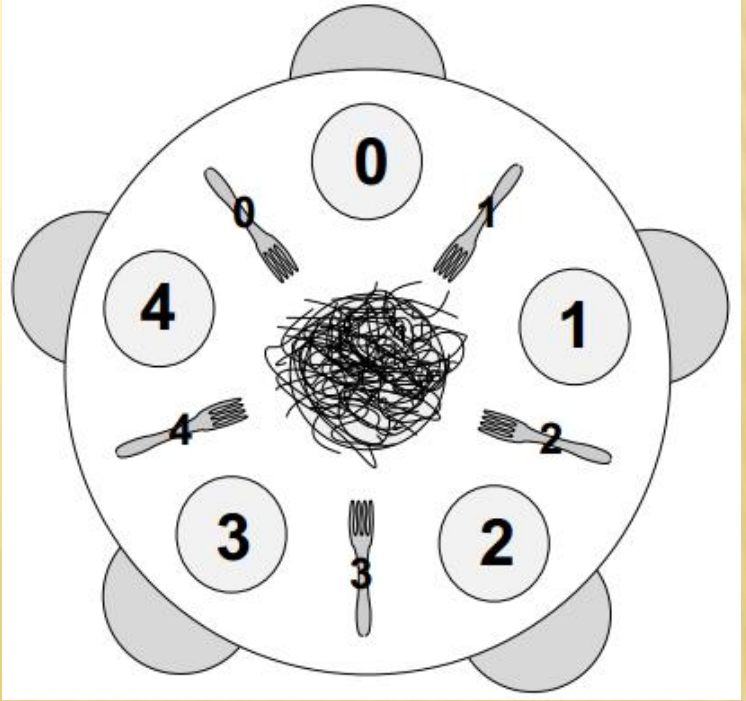
- Nepozname funkcie think() a eat(), ale:
  - think() nas z hladiska synchronizacie nijako nezaujima



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

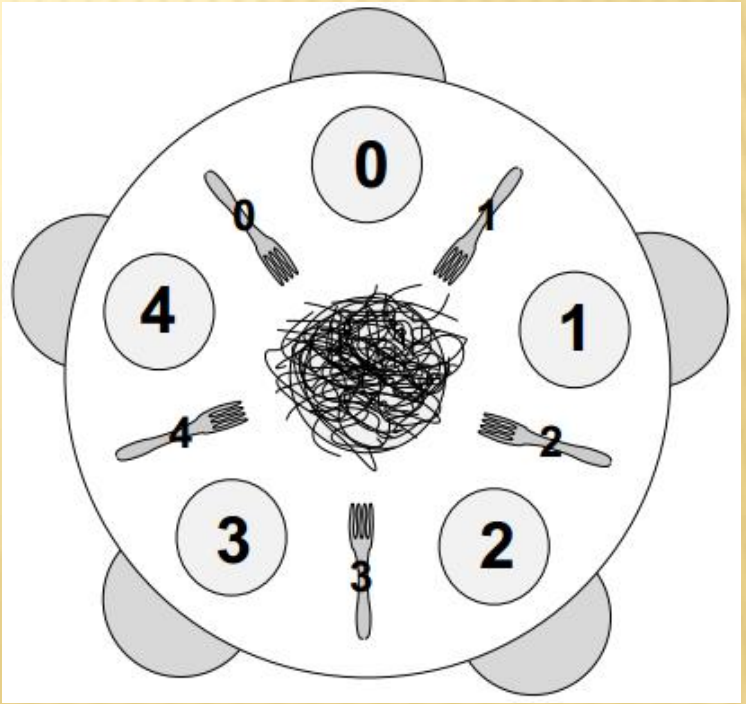
- Nepozname funkcie think() a eat(), ale:
  - think() nas z hladiska synchronizacie nijako nezaujima
  - eat() musi skoncit v konecnom case



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

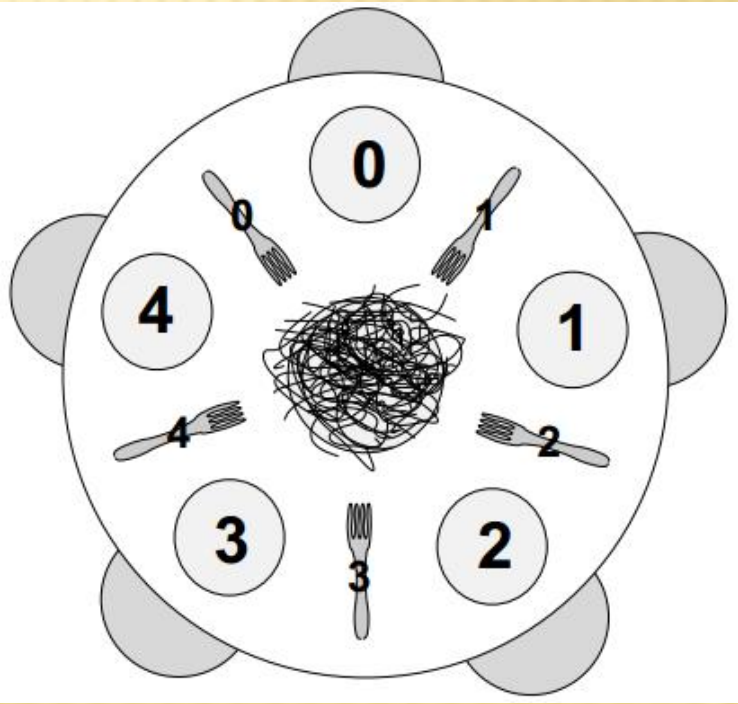
- Nepozname funkcie think() a eat(), ale:
  - think() nas z hladiska synchronizacie nijako nezaujima
  - eat() musi skoncit v konecnom case, inak nesplnime podmienku 3!



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

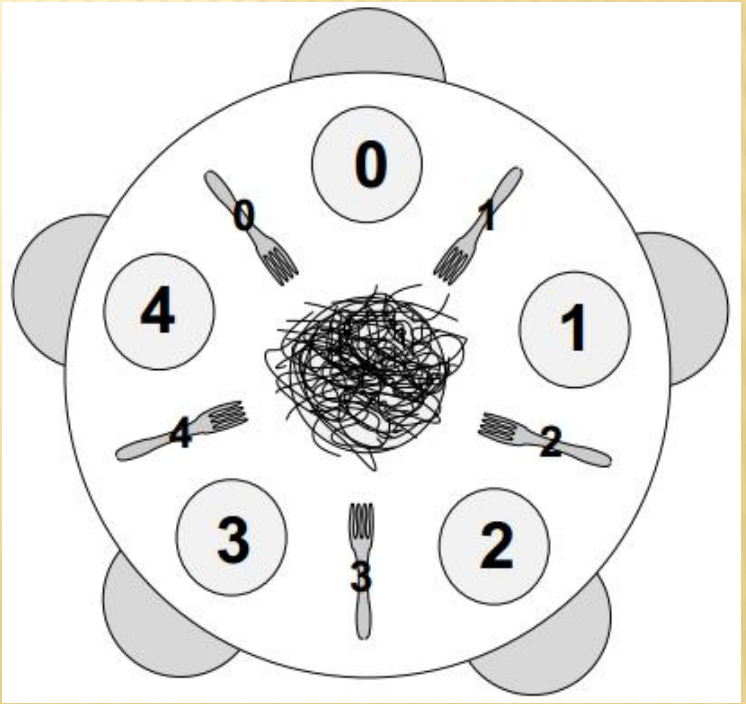
- Definujme si pomocne funkcie na urcenie id lavej a pravej vidlicky



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

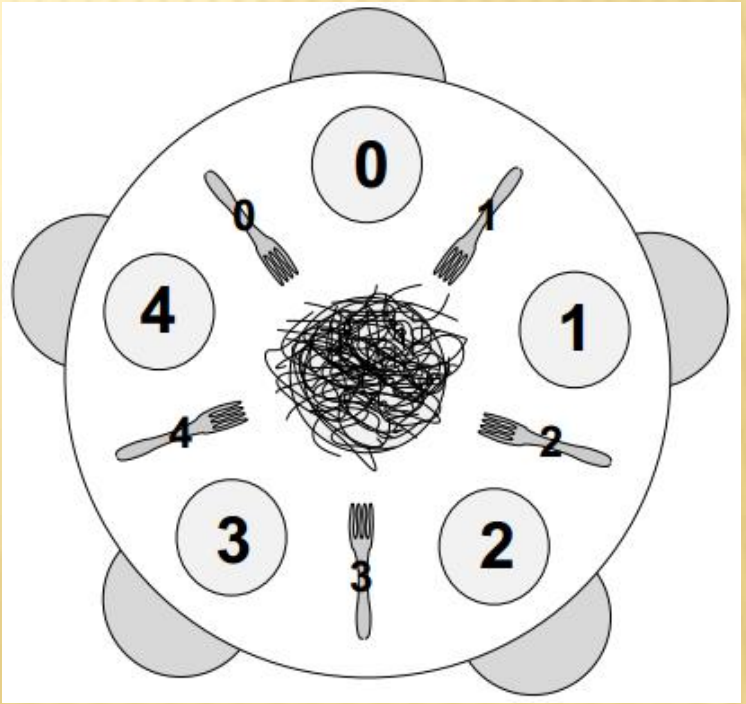
- Definujme si pomocne funkcie na urcenie id lavej a pravej vidlicky
- Kedze pre vidlicky ma platit exkluzivny pristup



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

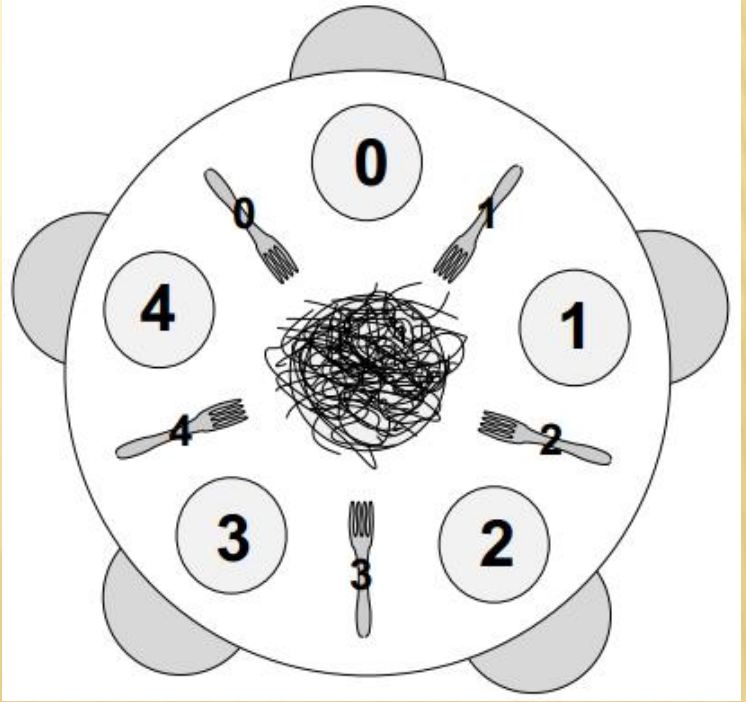
- Definujme si pomocne funkcie na urcenie id lavej a pravej vidlicky
- Kedze pre vidlicky ma platit exkluzivny pristup, pre kazdu z nich definujme binarny semafor



VECERAJUCI FILOZOFI

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

```
1    def right: return i
2    def left: return (i+1) % 5
3
4    forks = [Semaphore(1) for i in range(5)]
```



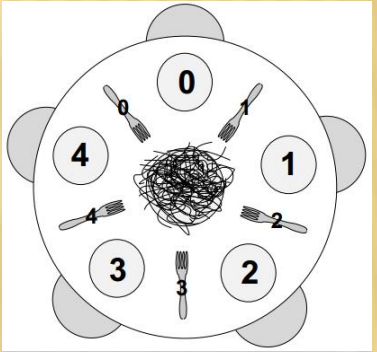
VECERAJUCI FILOZOFI #1

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

funkcia get_forks

```
1  def get_forks(i):
2      forks[right(i)].wait()
3      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```



VECERAJUCI FILOZOFI #1

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

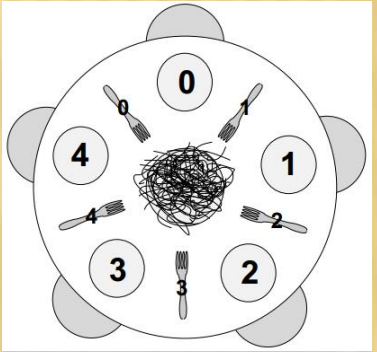
funkcia get_forks

```
1  def get_forks(i):
2      forks[right(i)].wait()
3      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

funkcia put_forks

```
1  def put_forks(i):
2      forks[right(i)].signal()
3      forks[left(i)].signal()
```



VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

funkcia get_forks

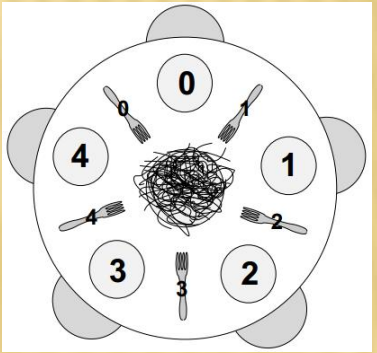
```
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

funkcia put_forks

```
1   def put_forks(i):
2       forks[right(i)].signal()
3       forks[left(i)].signal()
```



```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```



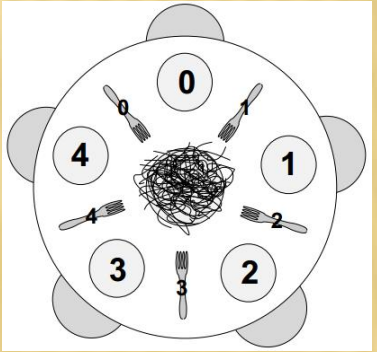
VECERAJUCI FILOZOFI #1

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      forks[right(i)].wait()
3      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Splname podmienku 1?



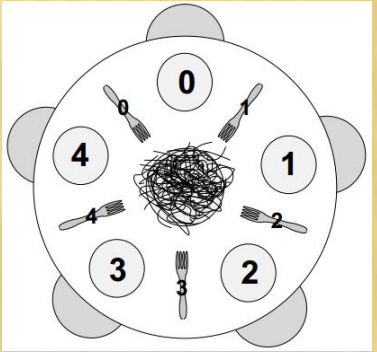
VECERAJUCI FILOZOFI #1

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      forks[right(i)].wait()
3      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Splname podmienku 1? (v 1 case drzi vidlicku 1 filozof)



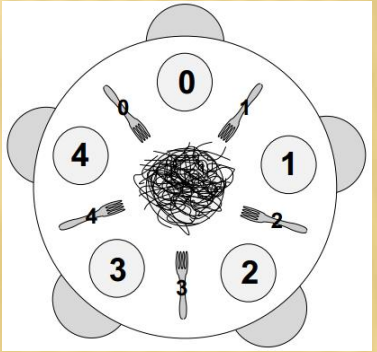
VECERAJUCI FILOZOFI #1

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      forks[right(i)].wait()
3      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Splname podmienku 1? (v 1 case drzi vidlicku 1 filozof)
- Splname podmienku 2?



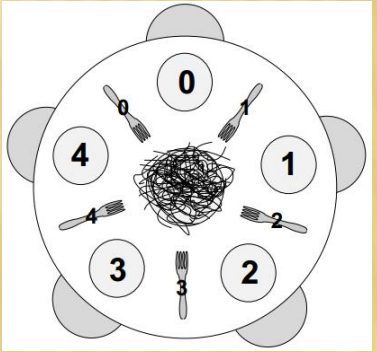
VECERAJUCI FILOZOFI #1

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      forks[right(i)].wait()
3      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Splname podmienku 1? (v 1 case drzi vidlicku 1 filozof)
- Splname podmienku 2? (nenastane uviaznutie)



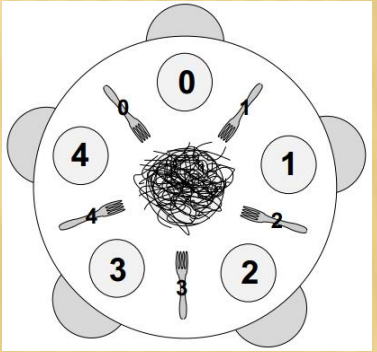
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Preco nastava uviaznutie?



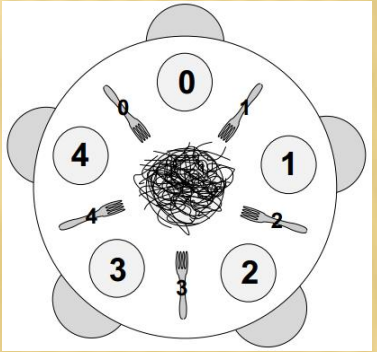
VECERAJUCI FILOZOFI #1

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      forks[right(i)].wait()
3      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Preco nastava uviaznutie?
  1. Okruhly stol



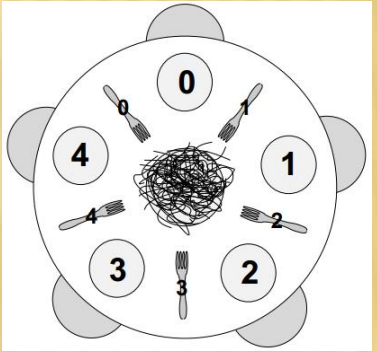
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Preco nastava uviaznutie?
  1. Okruhly stol
  2. Vsetci mozu sucasne jest



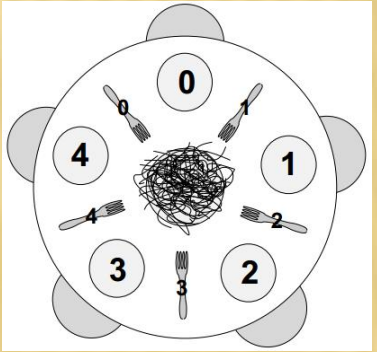
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Preco nastava uviaznutie?
  1. Okruhly stol
  2. Vsetci mozu sucasne jest
  3. Vsetci beru najprv pravu vidlicku



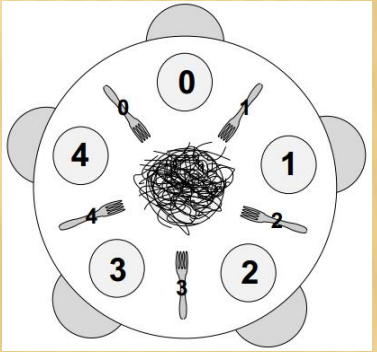
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?



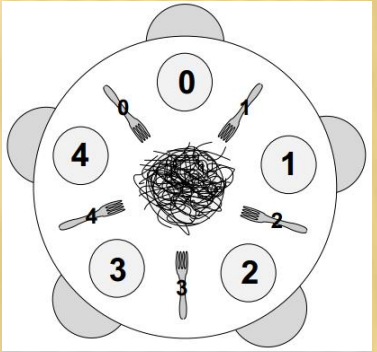
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?
- Zrusime jednu z podmienok uviaznutia



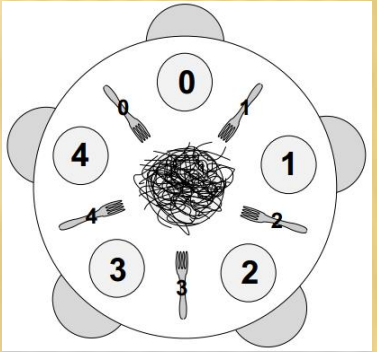
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?
  - Zmenime stol?



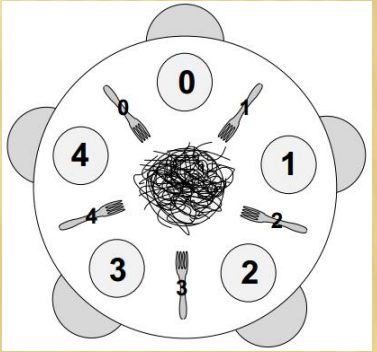
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?
  - Zmenime stol? Asi nie…



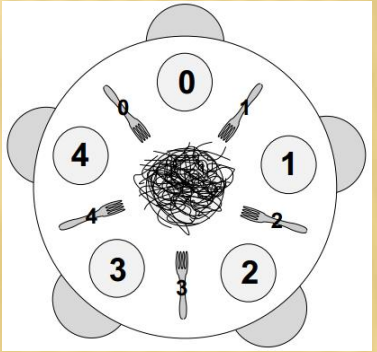
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?
  - Zmenime stol? Asi nie…
  - Umoznime jest naraz max 4?



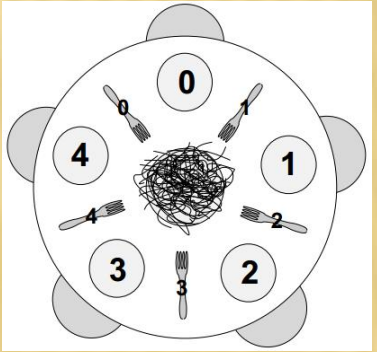
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?
  - Zmenime stol? Asi nie…
  - Umoznime jest naraz max 4? Ako?



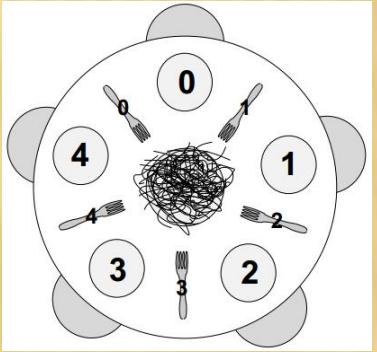
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?
  - Zmenime stol? Asi nie…
  - Umoznime jest naraz max 4? Ako?
  - Zavedieme lavakov a pravakov?



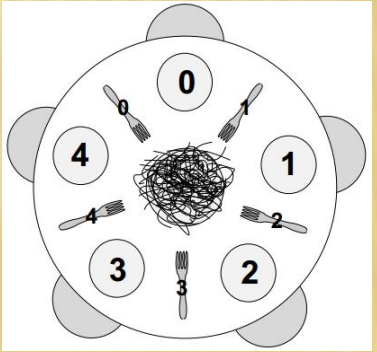
VECERAJUCI FILOZOFI #1

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?
  - Zmenime stol? Asi nie…
  - Umoznime jest naraz max 4? Ako?
  - Zavedieme lavakov a pravakov? Ako?



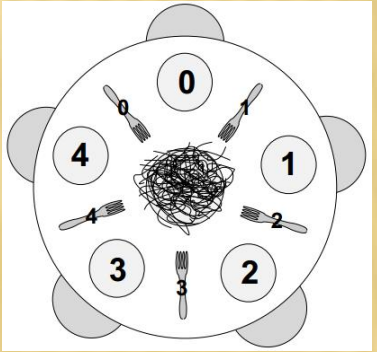
VECERAJUCI FILOZOFI #2

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
     funkcia get_forks
1    def get_forks(i):
2        forks[right(i)].wait()
3        forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Skusme riesit…



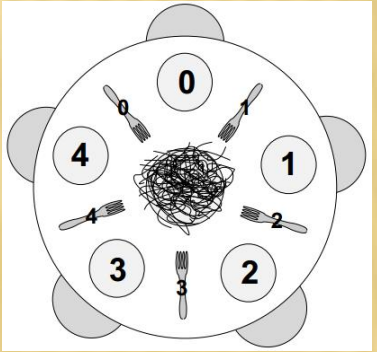
VECERAJUCI FILOZOFI #2

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Skusme riesit… obmedzenie max poctu tych, ktori mozu naraz jest



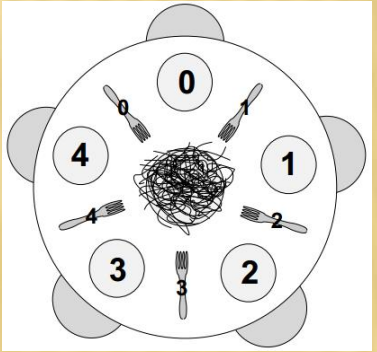
VECERAJUCI FILOZOFI #2

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Zavedieme casnika (footman)



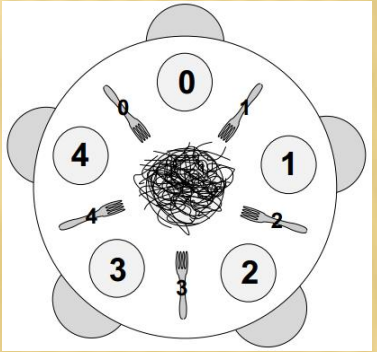
VECERAJUCI FILOZOFI #2

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Zavedieme casnika (footman)
  - + semafor



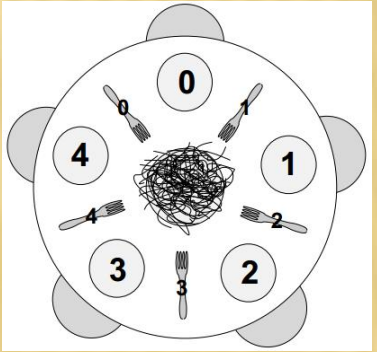
VECERAJUCI FILOZOFI #2

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Zavedieme casnika (footman)
  - semafor
  - inicializovany na max pocet naraz jeduvsich



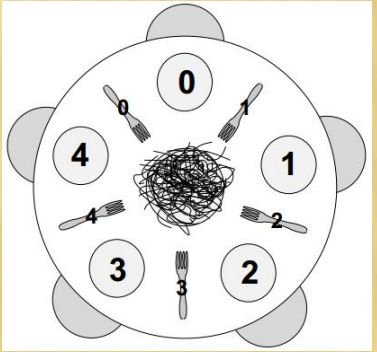
VECERAJUCI FILOZOFI #2

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
     funkcia get_forks
1    def get_forks(i):
2
3        forks[right(i)].wait()
4        forks[left(i)].wait()
```

```
1    while True:
2        think()
3        get_forks()
4        eat()
5        put_forks()
```

```
     funkcia put_forks
1    def put_forks(i):
2        forks[right(i)].signal()
3        forks[left(i)].signal()
4
```



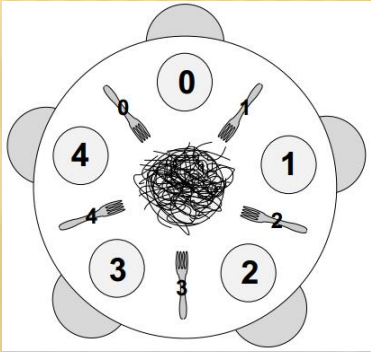
VECERAJUCI FILOZOFI #2

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      footman.wait()
3      forks[right(i)].wait()
4      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

```
   funkcia put_forks
1  def put_forks(i):
2      forks[right(i)].signal()
3      forks[left(i)].signal()
4      footman.signal()
```



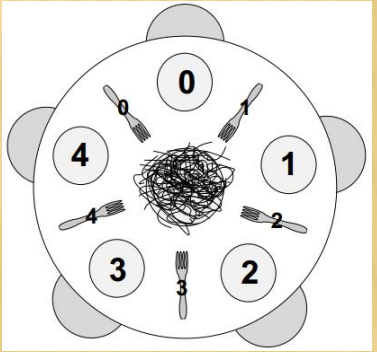
VECERAJUCI FILOZOFI #2

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      footman.wait()
3      forks[right(i)].wait()
4      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Podmienka 1 (1 v pre 1 f v 1 c)



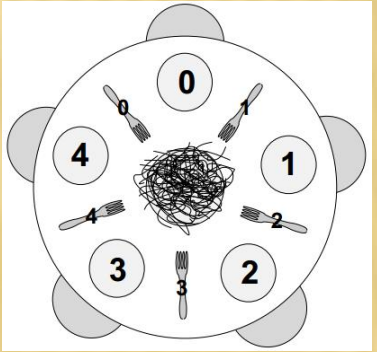
VECERAJUCI FILOZOFI #2

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      footman.wait()
3      forks[right(i)].wait()
4      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Podmienka 1 (1 v pre 1 f v 1 c)
- Podmienka 2 (deadlock)



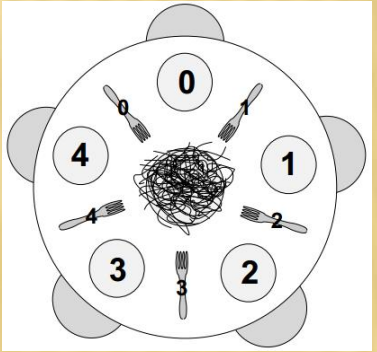
VECERAJUCI FILOZOFI #2

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      footman.wait()
3      forks[right(i)].wait()
4      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Podmienka 1 (1 v pre 1 f v 1 c)
- Podmienka 2 (deadlock)
- Podmienka 3 (vyhladnutie)



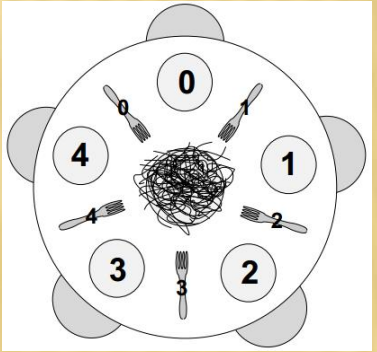
VECERAJUCI FILOZOFI #2

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       footman.wait()
3       forks[right(i)].wait()
4       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Podmienka 1 (1 v pre 1 f v 1 c)
- Podmienka 2 (deadlock)
- Podmienka 3 (vyhladnutie)
- Podmienka 4 (konkurentnost)



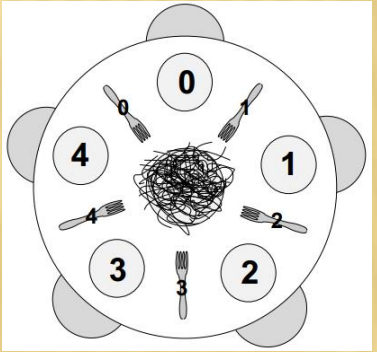
VECERAJUCI FILOZOFI #3

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Ako odstranime uviaznutie?
  - Zmenime stol? NIE
  - Umoznime jest naraz max 4? ANO
  - Zavedieme lavakov a pravakov? Ako?



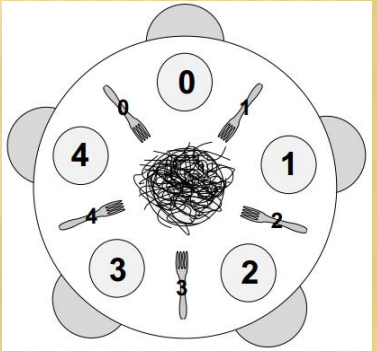
VECERAJUCI FILOZOFI #3

```
1  def right: return i
2  def left: return (i+1) % 5
3
4  forks = [Semaphore(1) for i in range(5)]
```

```
   funkcia get_forks
1  def get_forks(i):
2      forks[right(i)].wait()
3      forks[left(i)].wait()
```

```
1  while True:
2      think()
3      get_forks()
4      eat()
5      put_forks()
```

- Lavaci a pravaci



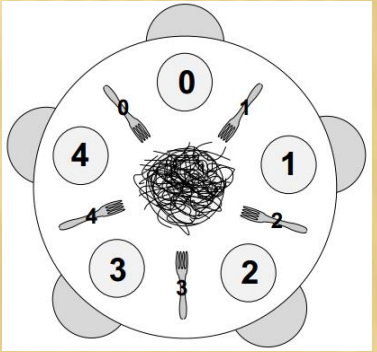
VECERAJUCI FILOZOFI #3

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Lavaci a pravaci
  - Aspon 1 lavak, aspon 1 pravak



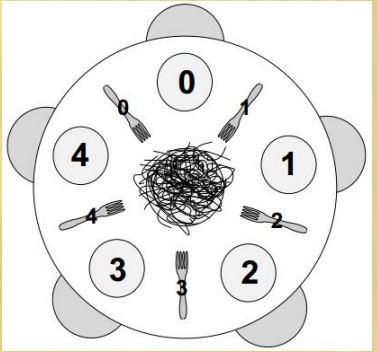
VECERAJUCI FILOZOFI #3

```
1   def right: return i
2   def left: return (i+1) % 5
3
4   forks = [Semaphore(1) for i in range(5)]
```

```
    funkcia get_forks
1   def get_forks(i):
2       forks[right(i)].wait()
3       forks[left(i)].wait()
```

```
1   while True:
2       think()
3       get_forks()
4       eat()
5       put_forks()
```

- Lavaci a pravaci
  - Aspon 1 lavak, aspon 1 pravak
  - Ako to funguje?



CVICENIE

- Prvá časť: obe riešenia problému filozofov
- Druhá časť: analýza a riešenie zápočtového príkladu z roku 2017