# Baboons crossing

Demonstrated by Ýmir Þórleifsson & Bjartur Þórhallsson

Explanation of synchronization scheme.

The scheme is simple. We need use a few variables to hold the current weight on the board, weight of a baboon, side of the baboon and the queue length. We also use a couple of constants, maxQueue and maxRopeWeight. Regarding semaphores, we use 3. One semaphore for each side and a rope semaphore.

When a baboon wants to cross the canyon, he starts by decrementing the rope sempahore and the mySide semaphore. This blocks all baboons from accessing the weighOnRope variable, and it also blocks baboons from the same side from continuing. Then the baboon accesses the rope, if the rope can support the baboon, he adds his wight to the rope. After that, we want our baboon to block the other side from going over the canyon. Lastly, he increments the rope and mySide samaphores. This lets all other baboons access the rope and baboons coming from the same side can continue. After swinging over the canyon, the baboon removes his weight from the rope and checks if he can let the other side go over the canyon. He can only do this if the rope is empty, and we have a queue (baboons waiting) on the other side. If this is true, he increments the otherSide semaphore and the other side can cross the canyon. After doing all this, he increments the rope semaphores, and lets all baboons access the weightOnRope variable.

Baboon and environment models

* Baboon

Each baboon holds a few variables, a side variable and a weight variable. This will be used to keep track of the total weight of the rope and to see which side the baboon is coming from.

* Environment

The only variable we need for the environment is the weightOnRope to hold the current weight that is on the rope. Baboons use this variable in order to check if the rope is at its maximum capacity. We use another constant, ropeMaxWeight (50kg), to compare weightOnRope to in order to perform the check for the rope’s maximum capacity.

Every time a baboon accesses the rope to cross the canyon, it adds its weight to the rope\_weight variable.

We also use this variable to see if there are no baboons currently on the rope. When a baboon is about to cross from one side and there are no baboons on the rope, the baboon needs to block the opposite direction from accessing the rope using the right\_side/left\_side semaphores.

Below is a rough implementation of the scheme. It does not perfectly, but we hope that this sheds some light on how our scheme works.

|  |
| --- |
| *package* main    *import* (  "fmt"  "math/rand"  "time"  )    *type* semaphore *chan* int    *const* maxRopeWeight = 50  *const* maxQueue = 5    *var* queueLength = 0  *var* weightOnRope = 0    // *Baboon is a function*  *func* *Baboon*(id int, mySide, otherSide, rope semaphore) {  *for* {  rope <- 0  mySide <- 0    // *Set random male or female, weight 20 ro 10 respectively*  *var* s []int  s = append(s, 10, 20)  myWeight := s[rand.Intn(2)]  queueLength++    // *Go if you can*  *for* {  *if* myWeight+weightOnRope <= maxRopeWeight {  weightOnRope += myWeight  *break*  }  }    // *Don't let the other side enter the rope!*  *if* queueLength > 0 {  otherSide <- 0  }    <-mySide  <-rope    // *Crossing the canyon (CS)*    // *Has crossed*  rope <- 0    weightOnRope -= myWeight  queueLength--    *if* weightOnRope == 0 || queueLength >= maxQueue {  queueLength = 0  mySide <- 0  <-otherSide  }  <-rope    fmt.Println(id, "crossing with weight:", weightOnRope)  }  }    *func* *main*() {  b1, b2, r := make(semaphore, 1), make(semaphore, 1), make(semaphore, 1)    fmt.Println("start")    *go* Baboon(1, b1, b2, r)  *go* Baboon(2, b2, b1, r)    time.Sleep(2 \* time.Second)  } |