

Tarea #2: Ejercicios de Sincronización

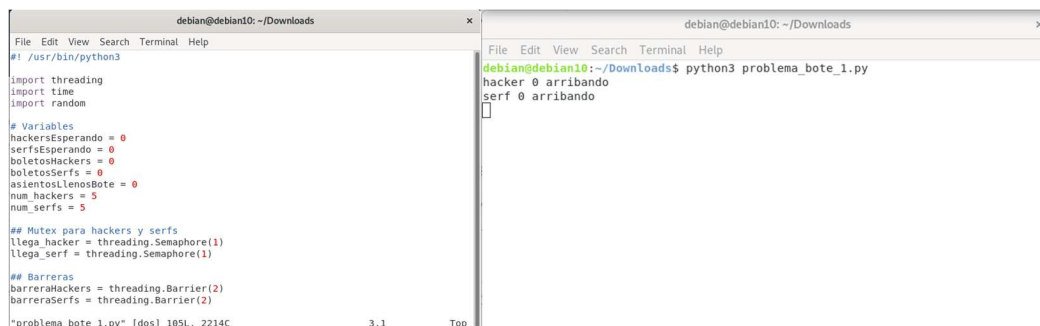
Problema elegido

Originalmente habíamos escogido el problema de los gatos y los ratones. No obstante, decidimos cambiar de problema y trabajar sobre el problema *del cruce del río*. El planteamiento es el siguiente:

- Para llegar a un encuentro de desarrolladores de sistemas operativos, hace falta cruzar un río en balsa.
- Los desarrolladores podrían pelearse entre sí, hay que cuidar que vayan con un balance adecuado.
- En la balsa caben cuatro (y sólo cuatro) personas.
 - o La balsa es demasiado ligera, y con menos de cuatro puede volcar.
- Al encuentro están invitados hackers (desarrolladores de Linux) y serfs (desarrolladores de Microsoft).
 - o Para evitar peleas, debe mantenerse un buen balance: No debes permitir que aborden tres hackers y un serf, o tres serfs y un hacker. Pueden subir cuatro del mismo bando, o dos y dos.
- Hay sólo una balsa.
- No se preocupen por devolver la balsa (está programada para volver sola)

Lenguaje y entorno

Para el desarrollo de esta tarea se utilizó el lenguaje de programación Python, corriendo las pruebas sobre Debian y utilizando vim para escribir el código. Para ejecutarlo solo es necesario bajar el código y ejecutarlo con la instrucción: *python3 problema_bote.py*



```
debian@debian10: ~/Downloads
File Edit View Search Terminal Help
#! /usr/bin/python3
import threading
import time
import random

# Variables
hackersEsperando = 0
serfsEsperando = 0
boletosHackers = 0
boletosSerfs = 0
asientosLlenosBote = 0
num_hackers = 5
num_serfs = 5

## Mutex para hackers y serfs
llega_hacker = threading.Semaphore(1)
llega_serf = threading.Semaphore(1)

## Barreras
barreraHackers = threading.Barrier(2)
barreraSerfs = threading.Barrier(2)

"problema_bote_1.py" [dos] 105L, 2214C 3,1 Top
```

```
debian@debian10: ~/Downloads
File Edit View Search Terminal Help
debian@debian10:~/Downloads$ python3 problema_bote_1.py
hacker 0 arribando
serf 0 arribando
```

Estrategia de sincronización

Primero aclarar que no pudimos generar un código 100% funcional. Intentamos utilizar diferentes patrones de sincronización. Específicamente usamos:

- Barrera: Quisimos utilizar barreras para poder limitar el paso de los hackers y los *serfs* en grupos de dos en dos. Lamentablemente, la implementación de las barreras en Python no tiene el método *signal*, solamente el método *reset* que reinicia la barrera.

Consideramos que el bug que limita el funcionamiento de nuestra implementación se encuentra justo en la sección que maneja la barrera. La señalización por cada *hacker/serf* debe ser individual, no un reseteo completo de la barrera.

- Semáforo: Utilizamos dos semáforos como *mutex* para poder bloquear el acceso a las variables que nos ayudan a hacer seguimiento al número de hackers en espera, los que ya tienen un boleto (un lugar apartado en el bote) y los asientos llenos en el bote.
- Variable de condición: También intentamos utilizar una variable de condición para poder dormir a cada hilo a la espera de que llegue la hora de zarpar. El último de ellos debería poder despertar al resto de hilos para que continúen sus procesos y “bajen” del barco.

Refinamientos

Este ejercicio no tiene refinamientos especificados.

Dudas

Tenemos la duda respecto a la forma correcta de utilizar la implementación de barreras en Python para lograr lo que establecimos anteriormente. También pensamos en que podría implementarse con semáforos, pero no encontramos como darle la vuelta al problema.

Además, pensamos que quizás nos complicamos demasiado con la solución y es probable que fuese más sencillo utilizar semáforos para todo, en lugar de las implementaciones propias del lenguaje.