PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
2da práctica (tipo a)
(Primer semestre de 2019)

Horario 0781: prof. V. Khlebnikov
Horario 0782: prof. A. Bello R.

Duración:	1 h. 50 min.
Nota:	No se puede usar ningún material de consulta.
	La presentación, la ortografía y la gramática influirán en la calificación.
Puntaje total:	20 puntos

Pregunta 1 **(6 puntos – 30 min.)**
a) (3 puntos – 15 min.) (***Ben-Ari, PCDP/2E***) Consider the following algorithm (cada sentencia `Pi` y `Qj` es atómica):

```
integer n ← 0

Hilo P                  Hilo Q

P1:  while n < 2        Q1:  n ← n + 1
P2:      write(n)       Q2:  n ← n + 1
```

Construct scenarios that give the output sequences: 012, 002, 02. (1 punto por TODOS los casos posibles)
Must the value 2 appear in the output? Explíquelo. (0,5 puntos)
How many times can the value 2 appear in the output? (0,5 puntos)
How many times can the value 1 appear in the output? (1 punto)

b) (3 puntos – 15 min.) Analice el siguiente pseudocódigo y determine si hay *race condition*:

```
volatile int lock;

/* i = 0,1 */
Hilo(i)

    lock=1-i
    while(lock);
    <sección crtica>
    lock=0
    <sección no crítica>
```

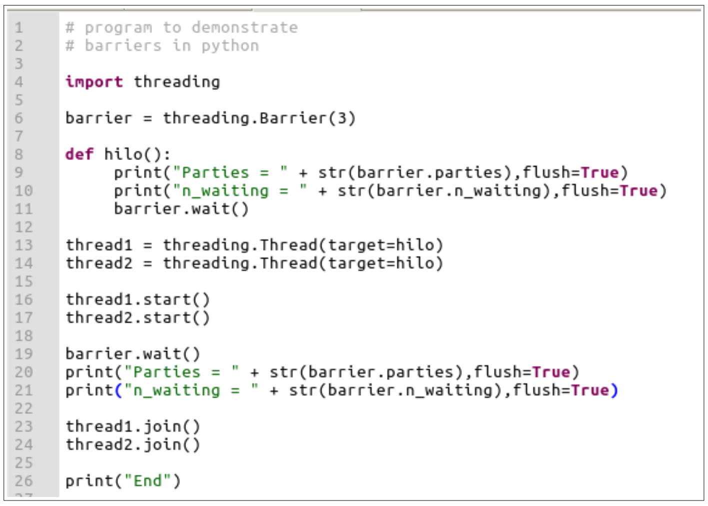
Pregunta 2 **(7 puntos – 35 min.)** (***Ben-Ari, PCDP/2E***) The following algorithm attempts to use binary semaphores to solve the critical section problem with **at most *k*** out of *N* processes in the critical section:

```
binary semaphore S ← 1,  delay ← 0
integer count ← k

Process P

    integer m
    loop forever
P1:     non-critical section
P2:     wait(S)
P3:     count ← count - 1
P4:     m ← count
P5:     signal(S)
P6:     if m < 0
P7:         wait(delay)
P8:     critical section
P9:     wait(S)
P10:    count ← count + 1
P11:    if count ≤ 0
P12:        signal(delay)
P13:    signal(S)
```

a) (4 puntos – 20 min.) For $N = 4$ and $k = 2$, construct a scenario in which `delay` = 2, contrary to the definition of a binary semaphore.
b) (3 puntos – 15 min.) Show this also for $N = 3$ and $k = 2$.

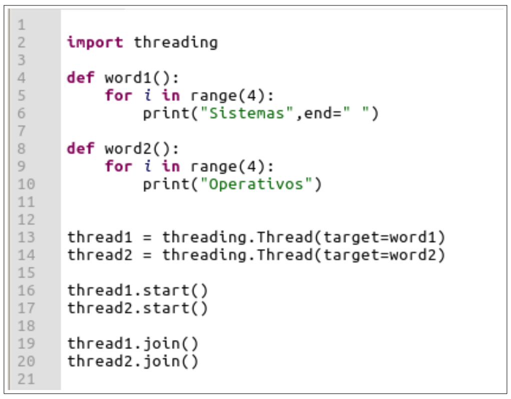
<u>Pregunta 3</u> (7 puntos – 35 min.) Barreras en Python (lea anexo).

a) (3 puntos – 15 min.) Dado el siguiente programa determine los posibles valores de `n_waiting` que cada hilo podría imprimir, explicando cada caso.



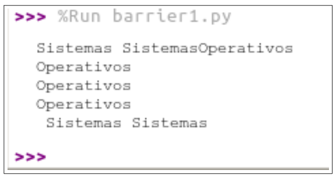
```python
# program to demonstrate
# barriers in python

import threading

barrier = threading.Barrier(3)

def hilo():
     print("Parties = " + str(barrier.parties),flush=True)
     print("n_waiting = " + str(barrier.n_waiting),flush=True)
     barrier.wait()

thread1 = threading.Thread(target=hilo)
thread2 = threading.Thread(target=hilo)

thread1.start()
thread2.start()

barrier.wait()
print("Parties = " + str(barrier.parties),flush=True)
print("n_waiting = " + str(barrier.n_waiting),flush=True)

thread1.join()
thread2.join()

print("End")
```

b) (4 puntos – 20 min.) Al ejecutar el siguiente programa

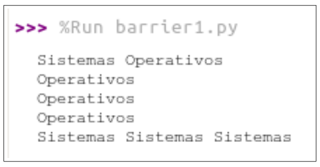


```python

import threading

def word1():
    for i in range(4):
        print("Sistemas",end=" ")

def word2():
    for i in range(4):
        print("Operativos")


thread1 = threading.Thread(target=word1)
thread2 = threading.Thread(target=word2)

thread1.start()
thread2.start()

thread1.join()
thread2.join()

```

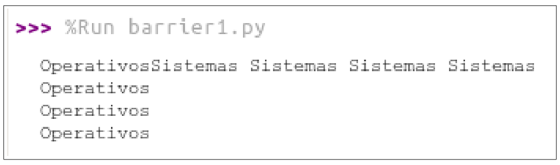
Se obtienen diferentes salidas, tales como:



```
>>> %Run barrier1.py

  Sistemas SistemasOperativos
  Operativos
  Operativos
  Operativos
   Sistemas Sistemas

>>>
```

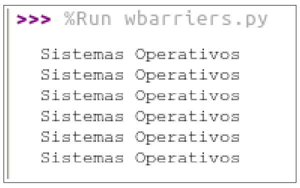


```
>>> %Run barrier1.py

  Sistemas Operativos
  Operativos
  Operativos
  Operativos
  Sistemas Sistemas Sistemas
```



```
>>> %Run barrier1.py

  OperativosSistemas Sistemas Sistemas Sistemas
  Operativos
  Operativos
  Operativos
```

Emplee barreras en cada hilo para que la salida siempre sea:



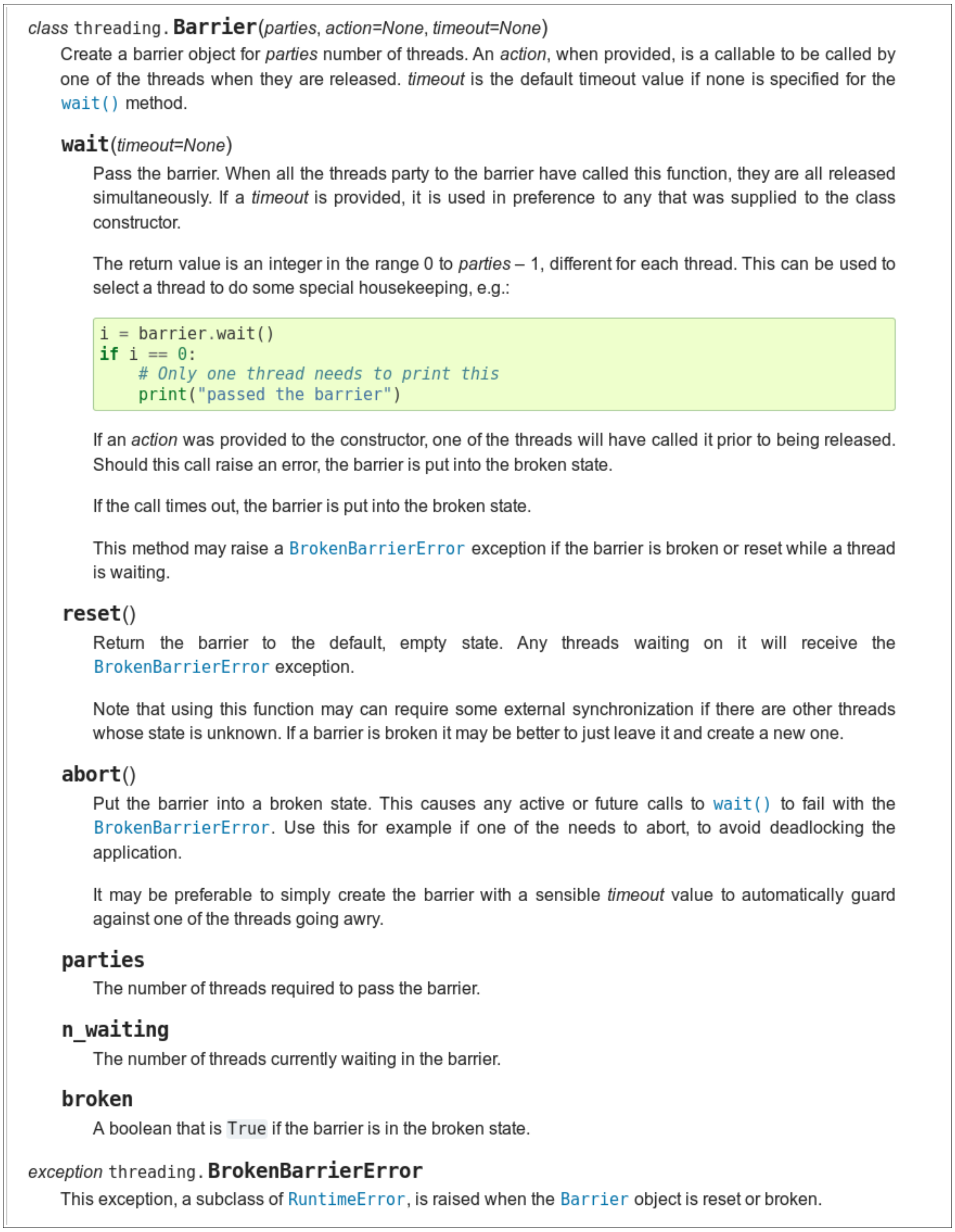
```
>>> %Run wbarriers.py

  Sistemas Operativos
  Sistemas Operativos
  Sistemas Operativos
  Sistemas Operativos
  Sistemas Operativos
  Sistemas Operativos
```

La práctica ha sido preparada por AB (1b,3) y VK (1a,2)
con LibreOffice Writer en Linux Mint 19.1 Tessa.

Profesor del curso: (0781) V. Khlebnikov
(0782) A. Bello R.

Pando, 26 de abril de 2019

ANEXOS



class `threading.`**`Barrier`**(*parties*, *action=None*, *timeout=None*)

Create a barrier object for *parties* number of threads. An *action*, when provided, is a callable to be called by one of the threads when they are released. *timeout* is the default timeout value if none is specified for the `wait()` method.

`wait`(*timeout=None*)

Pass the barrier. When all the threads party to the barrier have called this function, they are all released simultaneously. If a *timeout* is provided, it is used in preference to any that was supplied to the class constructor.

The return value is an integer in the range 0 to *parties* – 1, different for each thread. This can be used to select a thread to do some special housekeeping, e.g.:

```
i = barrier.wait()
if i == 0:
    # Only one thread needs to print this
    print("passed the barrier")
```

If an *action* was provided to the constructor, one of the threads will have called it prior to being released. Should this call raise an error, the barrier is put into the broken state.

If the call times out, the barrier is put into the broken state.

This method may raise a `BrokenBarrierError` exception if the barrier is broken or reset while a thread is waiting.

`reset`()

Return the barrier to the default, empty state. Any threads waiting on it will receive the `BrokenBarrierError` exception.

Note that using this function may can require some external synchronization if there are other threads whose state is unknown. If a barrier is broken it may be better to just leave it and create a new one.

`abort`()

Put the barrier into a broken state. This causes any active or future calls to `wait()` to fail with the `BrokenBarrierError`. Use this for example if one of the needs to abort, to avoid deadlocking the application.

It may be preferable to simply create the barrier with a sensible *timeout* value to automatically guard against one of the threads going awry.

`parties`

The number of threads required to pass the barrier.

`n_waiting`

The number of threads currently waiting in the barrier.

`broken`

A boolean that is `True` if the barrier is in the broken state.

exception `threading.`**`BrokenBarrierError`**

This exception, a subclass of `RuntimeError`, is raised when the `Barrier` object is reset or broken.

Tomado de: https://docs.python.org/3/library/threading.html