3. SINCRONIZAÇÃO

1. Operações com semáforos

A seguir, apresentamos uma sequência de operações do semáforo no início e no final das tarefas A, B, C. Considere que cada tarefa executa em um núcleo de processador dedicado. E considere que cada ação (P(Sx), V(Sx) ou .) possui tempo igual a 1T.

	Task A	Task B	Task C
1	P(SA)	P(SB)	P(SC)
2	P(SA)	.	P(SC)
3	P(SA)	.	P(SC)
4	.	.	.
5	.	.	.
6	.	V(SC)	V(SB)
7	V(SB)	V(SA)	V(SB)
8	END	.	V(SA)
9		END	END

Determine para os 6 casos a,b,c,d,e,f apresentados na tabela abaixo, se e em qual sequência as tarefas são executadas, usando as inicializações das variáveis do semáforo dadas na tabela.

Semáforos	a)	b)	c)	d)	e)	f)	g)
SA	2	3	2	0	3	1	1
SB	0	0	1	0	1	0	1
SC	2	2	1	3	3	3	1

A seguir, apresentamos uma nova sequência de operações do semáforo no início e no final das tarefas A, B, C. Considere que cada tarefa executa em um núcleo de processador dedicado. E considere que cada ação (P(Sx), V(Sx) ou .) possui tempo igual a 1T.

	Task A	Task B	Task C
1	P(SA)	P(SB)	P(SC)
2	P(SA)	P(SA)	P(SC)
3	V(SA)	.	P(SB)
4	.	.	.
5	.	.	.
6	.	P(SC)	V(SB)
7	V(SC)	V(SA)	V(SB)
8	END	END	V(SA)
9			END

Determine para os 6 casos a,b,c,d,e,f apresentados na tabela abaixo, se e em qual sequência as tarefas são executadas, usando as inicializações das variáveis do semáforo dadas na tabela.

Semáforos	a)	b)	c)	d)	e)	f)	g)
SA	2	1	2	0	3	2	1
SB	0	0	1	0	1	2	1
SC	2	1	1	2	1	2	1

2. Códigos em Python.

Nos seguintes códigos explique o comportamento do código e o conteúdo que será exibido ao final de sua execução.

```
#A
from threading import *
import time
l=Lock()
def wish(name,age):
  for i in range(3):
       l.acquire()
       print("Hi",name)
       time.sleep(2)
       print("Your age is",age)
       l.release()
t1=Thread(target=wish, args=("Sireesh",15))
t2=Thread(target=wish, args=("Nitya",20))
t1.start()
t2.start()
```

```
#B
from threading import *
import time
s=Semaphore(2)
def wish(name,age):
 for i in range(3):
      s.acquire()
      print("Hi",name)
      time.sleep(2)
      s.release()
t1=Thread(target=wish, args=("Sireesh",15))
t2=Thread(target=wish, args=("Nitya",20))
t3=Thread(target=wish, args=("Shiva",16))
t4=Thread(target=wish, args=("Ajay",25))
t1.start()
t2.start()
t3.start()
t4.start()
```

```
#C
from threading import Lock, Thread
lock = Lock()
g = 0

def add_one():
  global g
  lock.acquire()
  g += 1
  lock.release()

def add_two():
  global g
  lock.acquire()
  g += 2
  lock.release()

threads = []
for func in [add_one, add_two, add_two, add_one,
add_one, add_two]:
  threads.append(Thread(target=func))
  threads[-1].start()

for thread in threads:
  thread.join()

print(g)
```

3. Resolvendo problemas com Sincronização

A. A seguir é apresentado trecho de código Python. Análise o código e responda as seguintes questões:

I. Explique a finalidade do código apresentado?

II. Qual o resultado após execução do código?

III. Execute o código 10 vezes. Os resultados foram iguais? Caso negativo, por qual motivo?

IV. Utilize mecanismos de sincronização de forma que ao final da execução do código conta2 possua saldo 100 e conta1 possua saldo 0.

```
import threading
import time
class ContaBancaria():
 def __init__(self, nome, saldo):
   self.nome = nome
   self.saldo = saldo
 def __str__(self):
   return self.nome

conta1 = ContaBancaria("conta1", 100)
conta2 = ContaBancaria("conta2", 0)
class ThreadTransferenciaEntreContas(threading.Thread):
 def __init__(self, origem, destino, valor):
   threading.Thread.__init__(self)
   self.origem = origem
   self.destino = destino
   self.valor = valor

 def run(self):
   origem_saldo_inicial = self.origem.saldo
   origem_saldo_inicial -= self.valor
   time.sleep(0.001)
   self.origem.saldo = origem_saldo_inicial
   destino_saldo_inicial = self.destino.saldo
   destino_saldo_inicial += self.valor
   time.sleep(0.001)
   self.destino.saldo = destino_saldo_inicial
if __name__ == "__main__":

 threads = []
 for i in range(100):
   threads.append(ThreadTransferenciaEntreContas(conta1, conta2, 1))
 for thread in threads:
   thread.start()
 for thread in threads:
   thread.join()
 print('Saldo da', conta1, ':', conta1.saldo)
 print('Saldo da', conta2, ':', conta2.saldo)
```