

PROGRAMACIÓN CONCURRENTE

Práctica 4: Semáforos

Ejercicio 1. Dados los siguientes threads

```
thread {                                thread {
    print('A');                          print('E');
    print('B');                          print('F');
    print('C');                          print('G');
}
```

Utilizar semáforos para garantizar que, simultáneamente, 'A' se muestra antes que 'F', y 'F' se muestra antes que 'C'.

Ejercicio 2. Dados los siguientes threads:

```
thread {                                thread {
    print('C');                          print('A');
    print('E');                          print('R');
}
```

Utilizar semáforos para garantizar que las únicas salidas posibles sean ACERO y ACREO.

Ejercicio 3. Considerar los siguientes tres threads:

```
thread {                                thread {                                thread {
    print("R");                          print("I");                          print("O");
    print("OK");                         print("OK");                         print("OK");
}
```

Utilizar semáforos para garantizar que el único resultado impreso será R I O OK OK OK (asumimos que print es atómico).

Ejercicio 4. Dados los siguientes threads:

```
thread                                thread                                thread
while (true) {                        while (true) {                        while (true) {
    print('A');                        print('E');                          print('H');
    print('B');                        print('F');                          print('I');
    print('C');                        print('G');
    print('D');
}
```

Agregar semáforos para garantizar que simultáneamente se den las siguientes condiciones:

- La cantidad de 'F' es menor o igual a la cantidad de 'A'.
- La cantidad de 'H' es menor o igual a la cantidad de 'E'.

- La cantidad de 'C' es menor o igual a la cantidad de 'G'.

Ejercicio 5. Considere un programa compuesto por los siguientes tres threads:

```

thread T1: {
    x = x + 1;
}

thread T2: {
    x = x + 2;
}

thread T3: {
    x = x + 3;
}

```

Garantice, por medio del uso de semáforos, que luego de la ejecución del programa la variable x posea el valor 6. Procure que su solución no se bloquee incluso si sólo se ejecutaran dos (o uno) de los tres threads.

Ejercicio 6. Considere los siguientes threads que comparten dos variables y y z .

```

global int y = 0, z = 0;

thread {
    int x;
    x = y + z;
}

thread {
    y = 1;
    z = 2;
}

```

- ¿Cuáles son los posibles valores finales de x ?
- Para cada uno de los valores finales de x posibles, modifique el programa usando semáforos de forma tal que siempre x tenga ese valor al final de la ejecución (considere que el programa modificado siempre debe poder terminar).

Ejercicio 7. Considere los siguientes dos threads:

```

thread
    while (true)
        print('A');

thread
    while (true)
        print('B');

```

- Utilice semáforos para garantizar que en todo momento la cantidad de A y B difiera como máximo en 1.
- Modifique la solución para que la única salida posible sea ABABABABAB...

Ejercicio 8. Los siguientes threads cooperan para calcular el valor $N2$ que es la suma de los primeros N números impares. Los procesos comparten las variables N y $N2$ inicializadas de la siguiente manera: $N = 50$ y $N2 = 0$.

```

thread {
    while (N > 0)
        N = N - 1;
        print(N2);
}

thread
    while (true)
        N2 = N2 + 2 * N + 1;

```

Dé una solución que utilizando semáforos garantice que se muestra el valor correcto de $N2$.