UNIVERSIDAD TECNOLÓGICA DEL URUGUAY — **Programación 3**
INGENIERÍA MECATRÓNICA — **Segunda Evaluación Parcial**
ITR SUROESTE – Fray Bentos — Período **2023 · II**
Profesor: Giovani Bolzan Cogo — 24 de noviembre de 2023
Estudiante: _____ **CI:** _____ — **Tiempo límite:** 2h15'

Esta evaluación contiene 4 páginas y 6 cuestiones totalizando 17 puntos.

Material permitido: 1 hojas A4 de consulta.

Responder todo en 1 único archivo .PY, y correspondiente PDF, nombrado de la siguiente forma: `PRO3-nombre.apellido.CI******.py`, siendo "******" el correspondiente número de tu cédula de identidad.

Tabla de puntajes
(uso exclusivo docente)

Cuestión	Puntos	Resultado
1	1	
2	1	
3	2	
4	1	
5	4	
6	8	
Total:	17	

Calificación escala UTEC
(uso exclusivo docente)

1. (1 punto) Considera un entorno de procesamiento con una arquitectura de multiprocesadores. ¿Cuál es la diferencia entre paralelismo y concurrencia? Ejemplifica.

2. (1 punto) Considera un entorno de procesamiento multi hilos. ¿Qué funciones ejercen las estructuras de semáforos (**Semaphore**) y bloqueos (**Lock**) y cómo configurarlas desde el módulo `threading`?

3. (2 puntos) Considerando las estructuras de procesos e hilos, responde lo siguiente.
   (a) Utilización de recursos: Explora cómo los procesos e hilos aprovechan los recursos del sistema, como el sistema operativo, la memoria y el procesador.
   (b) Intercomunicación: Describe de manera concisa cómo los procesos e hilos se comunican entre sí en un sistema operativo.

4. (1 punto) Para un sistema de tareas de tiempo real, cita una estrategia de estructura de gestión de tareas. Ejemplifica describiendo brevemente una estructura conocida.

5. Considera un sistema que usa el módulo `threading` e implementa un código-fuente que soluciona los siguientes puntos.
   (a) (1 punto) Define una función `ejercicio5a` que utiliza un contador global y retorna el incremento del valor de ese contador. El valor a retornar es el contenido de contador.
   (b) (1 punto) Define una función `ejercicio5b` que utiliza una estrucutra de **Lock** para manipulación del contador global a traves de la llamada a la función desarrollada en `ejercicio5a`, así promoviendo una memoria protegida para manipulación.
   (c) (2 puntos) Define una función `ejercicio5c` que recibe un parámetro entero positivo n, ejecuta n hilos los cuales tengan como *target* la función anterior, `ejercicio5b`. El valor a retornar es el contenido de contador.

   Para este y para los siguientes ejercicios, considera el formato exacto de código establecido en Algoritmo 1. Los caracteres `@` deben ser sustituidos por código coherente.

Algoritmo 1: Estructura del código-fuente

```
"""
@author: <nombre del estudiante>
@exam: segunda evaluacion parcial
@course: PRO3-2023
"""
nombre = 'nombre.apellido'
def ejercicio5a():
    @ # definir funcionalidad #
    return respuesta
def ejercicio5b():
    @ # definir funcionalidad #
    return respuesta
def ejercicio5c(n):
    @ # definir funcionalidad #
    return respuesta
```

6. Considera los conceptos de semáforo, mutex y de hilos. Para los códigos-fuente en Algoritmos 2 a 4, implementa lo siguiente.

   (a) (2 puntos) define una clase **Competidor** la cual debe respetar el bosquejo del Algoritmo 2. Esta clase simula un competidor que participa de una competencia en una máquina de videojuego (*arcade*) y es, en realidad un hilo que se puede ejecutar.

Algoritmo 2: Clase Competidor

```
class Competidor( @ ): # debe heredar de una clase Thread #
    def __init__(self, nom, jug, pun=0):
        @ # debe llamar al constructor padre #
        self.nombre = @
        self.puntaje = @
        self.juguete = @
    def run(self):
        @ # debe realizar la accion de jugar #
```

   (b) (4 puntos) Una clase **Arcade** que tenga la funcionalidad de simular una máquina de juegos que permite 2 jugadores en simultáneo, como muestra el Algoritmo 3. Para eso se deber tener en cuenta que la máquina empieza con un número de fichas y se puede jugar mientras haya fichas disponibles para los competidores.

Algoritmo 3: Clase Arcade

```
class Arcade:
    def __init__(self, fic, s=2):
        @ # debe llamar al constructor padre #
        self.fichas = fic
        self.mutex = @ # debe definir la estructura mutex #
        self.semaphore = @ # debe definir la estructura semaforo #
    def jugar(self, competidor):
        while( @ ): # debe vrificar si hay fichas suficiente #

            @ # debe obtener habilitacion de mutex y semaforo #

            if self.fichas > 0:
                self.fichas -= 1
                print(f"{competidor.nombre} estah jugando. Quedan {
   self.fichas} fichas a disponibles.")
                sleep(randint(3)) # aguarda uso de maquina #
                competidor.puntaje += randint(2) # agrega puntos #
            else:
                print(f"No hay fichas disponibles para {competidor.
   nombre}.")
```

(c) (2 puntos) Una función principal **def** `main()` que hace uso de las clases `Arcade` y `Competidor` para que se pueda generar una instanciación de varios (n) jugadores, representados por hilos competidores, tal cual muestra el Algoritmo 4. La idea es generar un escenario simulado donde hay competidores que disputan por espacio en el*arcade* y juegan hasta que haya fichas disponibles. Al final se muestra un competidor ganador por el más alto atributo `puntaje`.

Algoritmo 4: código principal

```
def main(n):
    fichas, n_jugadores = 10, 2
    juego = Arcade( @ ) # define la plataforma de juego #
    competidores = @    # define la lista de competidores #

    @       # generar y ejecutar los hilos competidores #

    for competidor in competidores:
        print(f'{ @ } tiene { @ }puntos') # imprime info de competidor
    #
    ganador = max(competidores, key=lambda x:x.puntaje)
    print(f'ganador: {ganador.nombre}')
```

A modo de ejemplificación, el Algoritmo 5 muestra una ejecución del código-fuente completo (unión de los Algoritmos 2 a 4) con la configuración de los parámetros con 10 fichas, 5 jugadores totales y 2 jugadores en simultáneo.

Algoritmo 5: Ejemplo de ejecución del código

```
jugador_0 estah jugando. Quedan 9 fichas a disponibles.
jugador_1 estah jugando. Quedan 8 fichas a disponibles.
jugador_2 estah jugando. Quedan 7 fichas a disponibles.
jugador_3 estah jugando. Quedan 6 fichas a disponibles.
jugador_4 estah jugando. Quedan 5 fichas a disponibles.
jugador_0 estah jugando. Quedan 4 fichas a disponibles.
jugador_1 estah jugando. Quedan 3 fichas a disponibles.
jugador_2 estah jugando. Quedan 2 fichas a disponibles.
jugador_3 estah jugando. Quedan 1 fichas a disponibles.
jugador_4 estah jugando. Quedan 0 fichas a disponibles.
No hay fichas disponibles para jugador_0.
No hay fichas disponibles para jugador_1.
No hay fichas disponibles para jugador_2.
No hay fichas disponibles para jugador_3.
jugador_0 tiene 0 puntos
jugador_1 tiene 1 puntos
jugador_2 tiene 0 puntos
jugador_3 tiene 1 puntos
jugador_4 tiene 2 puntos
ganador: jugador_4
```