UNIVERSIDAD RAFAEL LANDÍVAR

FACULTAD DE INGENIERÍA

SISTEMAS OPERATIVOS

SECCIÓN 1 VESPERTINA

ING. JULIO REQUENA

EJERCICIOS MONITORES

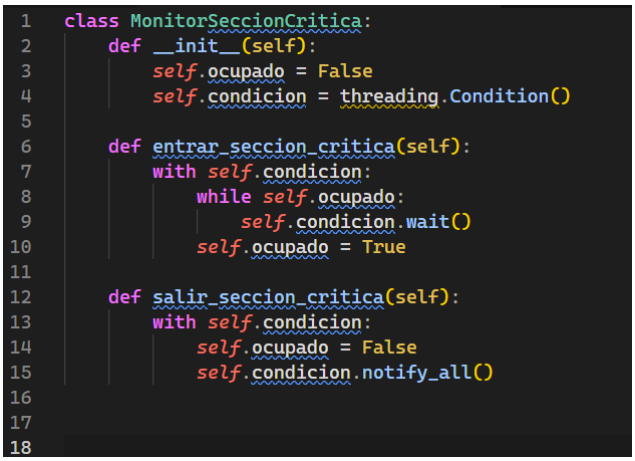
Julio Anthony Engels Ruiz Coto 1284719

GUATEMALA DE LA ASUNCIÓN, MARZO 23 DE 2023

CAMPUS CENTRAL

I. MONITORES EN SISTEMAS OPERATIVOS

En sistemas operativos se conoce como monitor a mecanismos de sincronización que se utiliza para gestionar el acceso concurrente a recursos compartidos por múltiples hilos o procesos de ejecución. Los monitores facilitan la implementación de soluciones a problemas de concurrencia por ejemplo la exclusión mutua, la sincronización de procesos y la comunicación entre procesos. A menudo los monitores se implementan utilizando primitivas de sincronización de bajo nivel como por ejemplo los semáforos.

EJERCICIO 1	Control de acceso a una sección critica: El ejercicio consiste que se tienen varios procesos que deben acceder a una sección critica, pero solo un proceso puede acceder a la vez. 
EJERCICIO 2	Productor y consumidores: El ejercicio consiste en que se tienen varios procesos productores y consumidores que comparten un buffer con una capacidad limitada.

```python
class MonitorSeccionCritica:
    def __init__(self):
        self.ocupado = False
        self.condicion = threading.Condition()

    def entrar_seccion_critica(self):
        with self.condicion:
            while self.ocupado:
                self.condicion.wait()
            self.ocupado = True

    def salir_seccion_critica(self):
        with self.condicion:
            self.ocupado = False
            self.condicion.notify_all()


```

```python
class MonitorProductoresConsumidores:
    def __init__(self, buffer_size):
        self.buffer = []
        self.buffer_size = buffer_size
        self.condicion = threading.Condition()

    def producir(self, item):
        with self.condicion:
            while len(self.buffer) == self.buffer_size:
                self.condicion.wait()
            self.buffer.append(item)
            self.condicion.notify_all()

    def consumir(self):
        with self.condicion:
            while not self.buffer:
                self.condicion.wait()
            item = self.buffer.pop(0)
            self.condicion.notify_all()
        return item

```

EJERCICIO 3

Lectores y Escritores:

El ejercicio consiste en que se tienen varios procesos de lectores y escritores que comparten un recurso para los lectores pueden acceder simultáneamente al recurso, pero para los escritores deben tener acceso exclusivo.

```python
class MonitorLectoresEscritores:
    def __init__(self):
        self.num_lectores = 0
        self.escribiendo = False
        self.condicion = threading.Condition()

    def leer(self):
        with self.condicion:
            while self.escribiendo:
                self.condicion.wait()
            self.num_lectores += 1

        # Leer recurso aquí
        print("Leyendo ...")

        with self.condicion:
            self.num_lectores -= 1
            self.condicion.notify_all()

    def escribir(self):
        with self.condicion:
            while self.escribiendo or self.num_lectores > 0:
                self.condicion.wait()
            self.escribiendo = True

        # Escribir en recurso aquí
        print("Escribiendo ...")

        with self.condicion:
            self.escribiendo = False
            self.condicion.notify_all()
```