

Universidad Nacional Autónoma de México

Facultad de Ingeniería

Sistemas operativos.

Semestre 2024-1

Profesor:

Ing. Gunnar Eyal Wolf Iszaevich

Grupo: 06

Alumno: Nuñez Rodas Abraham Enrique

Proyecto 02



Índice.

| | |
|--|----------|
| Descripción de los mecanismos de sincronización empleados | 2 |
| Lógica de operación: | 3 |
| Identificación del estado compartido: | 3 |
| Descripción algorítmica del avance de cada hilo/proceso:..... | 4 |
| Descripción de la interacción entre ellos:..... | 4 |
| Entorno de Desarrollo: | 5 |
| Lenguaje de Programación:..... | 5 |
| Bibliotecas:..... | 5 |
| Requisitos para la Ejecución: | 6 |
| Ejemplos o pantallazos de una ejecución exitosa | 7 |

Descripción de los mecanismos de sincronización empleados

1. Lock (Mutex):

- Uso en el código: `'self.lock_equipamiento'`
- Descripción: Un Lock o Mutex es un mecanismo de sincronización fundamental que se utiliza para controlar el acceso exclusivo a un recurso compartido. En este caso, se usa para asegurar que solo un chef (hilo) pueda utilizar el equipamiento de la cocina a la vez, evitando condiciones de carrera y posibles conflictos.

2. Semaphore:

- Uso en el código: `'self.sem_meseros'`
- Descripción: Un semáforo es un contador que controla el acceso a un recurso que tiene una capacidad limitada. En la simulación, se utiliza para restringir el número de meseros (hilos) que pueden servir platos al mismo tiempo, lo que refleja la limitación de capacidad en el entorno de un restaurante real.

3. Condition Variable:

- Uso en el código: `'self.cond_plato_listo'`
- Descripción: Una variable de condición se utiliza para bloquear un hilo hasta que se cumpla cierta condición. En este código, se emplea para hacer que los meseros esperen (bloqueados) hasta que se les notifique que un plato está listo para ser servido. Esto asegura que los meseros no intenten servir pedidos antes de que los chefs hayan terminado de prepararlos.

4. Queue:

- Uso en el código: `'self.pedidos'`
- Descripción: La cola es una estructura de datos de hilo seguro que se usa para almacenar los pedidos que los chefs deben procesar. Asegura que los pedidos se manejen en el orden

en que llegan (FIFO - First In, First Out) y que no haya condiciones de carrera al acceder a los pedidos.

5. Flag (Bandera):

- Uso en el código: `'self.todos_pedidos_servidos'`

- Descripción: Se introduce una bandera como una variable booleana que indica si todos los pedidos han sido servidos. Se utiliza en combinación con una variable de condición para señalar a los meseros cuando todos los platos han sido servidos y pueden dejar de esperar por más platos para servir.

Estos mecanismos de sincronización trabajan juntos para coordinar la interacción entre chefs y meseros en la simulación, asegurando que el trabajo se realice de manera ordenada y sin conflictos.

Lógica de operación:

La lógica de operación del código simula un entorno de restaurante donde múltiples chefs preparan pedidos y múltiples meseros sirven los platos preparados. Los pedidos llegan de manera asíncrona y son puestos en una cola. Los chefs toman pedidos de esta cola para prepararlos uno a la vez, mientras que los meseros esperan a que los pedidos estén listos para servirlos a los clientes.

Identificación del estado compartido:

1. Cola de Pedidos (`'self.pedidos'`): Es una estructura de datos compartida que almacena los pedidos que deben ser preparados por los chefs. Es accesible por todos los hilos de chefs y se asegura de manejarlos de forma segura en un entorno concurrente.

2. Lock de Equipamiento (`'self.lock_equipamiento'`): Controla el acceso al equipamiento de cocina, asegurando que solo un chef a la vez pueda usarlo.

3. Semáforo de Meseros (`'self.sem_meseros'`): Limita la cantidad de meseros que pueden servir platos simultáneamente.

4. Variable de Condición ('self.cond_plato_listo'): Permite que los meseros esperen de manera eficiente la señal de que hay platos listos para ser servidos
5. Bandera de Pedidos Servidos ('self.todos_pedidos_servidos'): Indica cuándo todos los pedidos han sido servidos para que los meseros puedan detener su ejecución.

Descripción algorítmica del avance de cada hilo/proceso:

- Chefs (Hilos de Chefs):

1. Esperan a que haya un pedido disponible en la cola.
2. Tomar un pedido y adquirir el lock de equipamiento.
3. Preparar el pedido simulando un tiempo de preparación.
4. Liberar el lock de equipamiento y notificar a los meseros que un pedido está listo.
5. Marcar el pedido como completado en la cola.

- Meseros (Hilos de Meseros):

1. Esperar a que se notifique que un plato está listo.
2. Comprobar la bandera de pedidos servidos; si todos están servidos, el hilo se detiene.
3. Adquirir un semáforo para proceder a servir.
4. Simular la acción de servir un plato.
5. Liberar el semáforo para que otro mesero pueda servir.

Descripción de la interacción entre ellos:

- Los chefs trabajan de forma independiente entre sí gracias al uso del lock de equipamiento que asegura que solo un chef a la vez puede preparar un pedido.

- Los meseros son notificados mediante la variable de condición cada vez que un pedido está listo. Esto crea un punto de sincronización donde los meseros se bloquean y esperan eficientemente sin consumir CPU.
- El semáforo gestiona el número de meseros que pueden servir platos al mismo tiempo, previniendo el exceso de concurrencia en la etapa de servicio.
- La bandera 'self.todos_pedidos_servidos' asegura que los meseros no continúen esperando por platos para servir una vez que todos los pedidos han sido manejados, permitiendo que el programa termine de manera ordenada y sin hilos colgados.

Entorno de Desarrollo:

Para reproducir una ejecución exitosa del programa de simulación de la cocina de restaurante, se necesita configurar el siguiente entorno de desarrollo:

Lenguaje de Programación:

Lenguaje: Python

Versión: La versión utilizada debe ser Python 3.6 o superior, ya que utiliza características de sincronización de hilos que están disponibles en estas versiones.

Bibliotecas:

Estándar del Lenguaje: El código utiliza bibliotecas que son parte de la biblioteca estándar de Python, no se requieren instalaciones adicionales de terceros.

threading: Para el manejo de hilos.

queue: Para estructuras de datos seguras en hilos.

time: Para las funciones de tiempo y espera.

random: Para generar tiempos de espera aleatorios que simulan la preparación y el servicio.

Sistema Operativo/Distribución:

Compatibilidad: El código es independiente del sistema operativo y debería funcionar en cualquier distribución que soporte la versión de Python mencionada. Esto incluye sistemas operativos como Windows, macOS y Linux.

Desarrollo y Pruebas: Aunque no se especifica, el código puede haber sido desarrollado y probado en cualquier sistema operativo común. Por ejemplo, podría ser cualquier versión reciente de Windows (Windows 10/11), macOS (macOS 10.15 Catalina o superior), o una distribución de Linux como Ubuntu (Ubuntu 18.04 LTS o superior).

Requisitos para la Ejecución:

Tener instalada la versión adecuada de Python.

Contar con un entorno que permita la ejecución de scripts de Python, como la terminal en macOS/Linux o el Command Prompt/PowerShell en Windows.

No es necesario ningún IDE específico, pero se puede utilizar uno como PyCharm, VSCode, o incluso un editor de texto simple, para editar y ejecutar el código.

Acceso a una terminal o línea de comandos para ejecutar el script.

Con esta configuración, cualquier usuario debería ser capaz de reproducir la ejecución del programa sin problemas adicionales.

Ejemplos o pantallazos de una ejecución exitosa

Código con num_chefs = 3 num_meseros = 2

```
import threading
import queue
import time
import random

# Simulación de una cocina de restaurante

class Restaurante:
    def __init__(self, num_chefs, num_meseros):
        self.pedidos = queue.Queue()
        self.lock_equipamiento = threading.Lock()
        self.sem_meseros = threading.Semaphore(num_meseros)
        self.cond_plato_listo = threading.Condition()
        self.todos_pedidos_servidos = False # Agregada variable para controlar el flujo

    def chef(self, id_chef):
        while True:
            pedido = self.pedidos.get()
            if pedido is None: # Si se recibe None, es la señal para detenerse
                self.pedidos.task_done()
                break
            with self.lock_equipamiento:
                print(f"Chef {id_chef} está preparando el pedido {pedido}")
                time.sleep(random.uniform(0.5, 2))
            with self.cond_plato_listo:
                print(f"Chef {id_chef} ha terminado el pedido {pedido}")
                self.cond_plato_listo.notify_all()
                self.pedidos.task_done()

    def mesero(self, id_mesero):
        while True:
            with self.cond_plato_listo:
                self.cond_plato_listo.wait()
                if self.todos_pedidos_servidos: # Verificar si ya se sirvieron todos los pedidos
                    break
            with self.sem_meseros:
                print(f"Mesero {id_mesero} está sirviendo un plato.")
                time.sleep(random.uniform(0.5, 1))

    def nuevo_pedido(self, id_pedido):
        print(f"Recibido nuevo pedido: {id_pedido}")
        self.pedidos.put(id_pedido)

    def iniciar_servicio(self, num_pedidos):
        # Iniciar hilos de chefs
        for i in range(num_chefs):
            threading.Thread(target=self.chef, args=(i,)).start()

        # Iniciar hilos de meseros
        for i in range(num_meseros):
            threading.Thread(target=self.mesero, args=(i,)).start()

        # Simular llegada de pedidos
        for i in range(num_pedidos):
            time.sleep(random.uniform(0.1, 0.3))
            self.nuevo_pedido(i)

        # Colocar None en la cola para cada chef como señal de parada
        for i in range(num_chefs):
            self.pedidos.put(None)

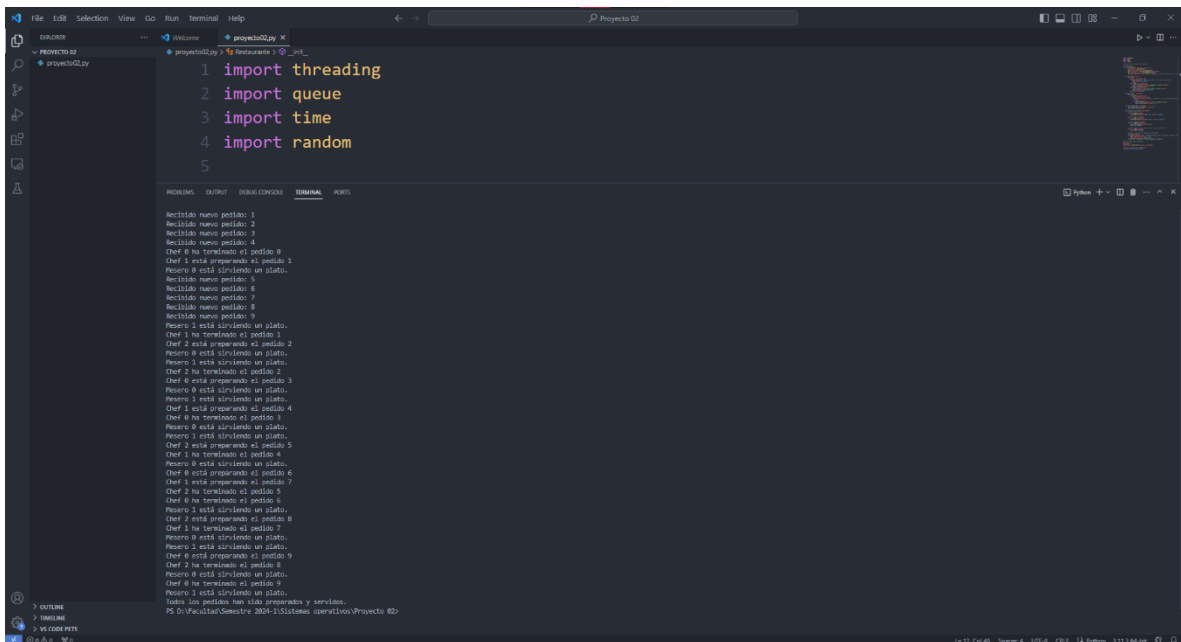
        # Esperar a que todos los pedidos sean procesados
        self.pedidos.join()
        with self.cond_plato_listo: # Asegurarse de que todos los meseros revisen la condición de salida
            self.todos_pedidos_servidos = True
            self.cond_plato_listo.notify_all()
        print("Todos los pedidos han sido preparados y servidos.")

# Crear instancia del restaurante
num_chefs = 3
num_meseros = 2
restaurante = Restaurante(num_chefs, num_meseros)

# Iniciar el servicio del restaurante
restaurante.iniciar_servicio(10)

}
```

Pantallazo de ejecución exitosa.



```
1 import threading
2 import queue
3 import time
4 import random
5

Recibido nuevo pedido: 1
Recibido nuevo pedido: 2
Recibido nuevo pedido: 3
Recibido nuevo pedido: 4
Chef 0 ha terminado el pedido 0
Chef 3 está preparando el pedido 1
Mesero 0 está sirviendo un plato.
Recibido nuevo pedido: 5
Recibido nuevo pedido: 6
Recibido nuevo pedido: 7
Recibido nuevo pedido: 8
Recibido nuevo pedido: 9
Mesero 3 está sirviendo un plato.
Chef 1 ha terminado el pedido 1
Chef 2 ha terminado el pedido 2
Mesero 0 está sirviendo un plato.
Mesero 1 está sirviendo un plato.
Chef 2 ha terminado el pedido 2
Chef 0 está preparando el pedido 3
Mesero 0 está sirviendo un plato.
Mesero 1 está sirviendo un plato.
Chef 1 está preparando el pedido 4
Chef 0 ha terminado el pedido 3
Mesero 0 está sirviendo un plato.
Mesero 3 está sirviendo un plato.
Chef 2 está preparando el pedido 4
Chef 1 está preparando el pedido 5
Chef 0 ha terminado el pedido 4
Chef 2 ha terminado el pedido 5
Chef 0 ha terminado el pedido 5
Mesero 2 está sirviendo un plato.
Chef 2 está preparando el pedido 6
Chef 1 ha terminado el pedido 7
Mesero 0 está sirviendo un plato.
Mesero 3 está sirviendo un plato.
Chef 0 está preparando el pedido 6
Chef 2 ha terminado el pedido 6
Mesero 0 está sirviendo un plato.
Mesero 3 está sirviendo un plato.
Chef 0 ha terminado el pedido 6
Mesero 0 está sirviendo un plato.
Chef 0 ha terminado el pedido 7
Mesero 3 está sirviendo un plato.
Todos los pedidos han sido preparados y servidos.
PS D:\Facultad\Semestre 2004-5\Sistemas operativos\Proyecto 02>
```

Se intentó hacer la interfaz gráfica, pero no resultó exitosa, ya que está terminada por no responder y no terminar de procesar a comparación del código sin interfaz, sin embargo, hasta aquí está el resultado hasta donde pude llegar.

Código con num_chefs = 3 num_meseros = 2:

```
import tkinter as tk
import threading
import queue
import time
import random

# Simulación de una cocina de restaurante
class Restaurante:
    def __init__(self, num_chefs, num_meseros, gui_ref=None):
        self.num_chefs = num_chefs # Guarda num_chefs como variable de instancia
        self.num_meseros = num_meseros
        self.pedidos = queue.Queue()
        self.lock_equipamiento = threading.Lock()
        self.sem_meseros = threading.Semaphore(num_meseros)
        self.cond_plato_listo = threading.Condition()
        self.cond_mesero = threading.Condition() # Nueva variable de condición para coordinar con los
        self.todos_pedidos_servidos = False
        self.gui_ref = gui_ref # Referencia a la GUI para actualizaciones

    def chef(self, id_chef):
        while True:
            pedido = self.pedidos.get()
            if pedido is None: # Si se recibe None, es la señal para detenerse
                self.pedidos.task_done()
                break
            with self.lock_equipamiento:
                self.actualizar_gui(f"Chef {id_chef} está preparando el pedido {pedido}")
                time.sleep(random.uniform(0.5, 2))
            with self.cond_plato_listo:
                self.actualizar_gui(f"Chef {id_chef} ha terminado el pedido {pedido}")
                self.cond_plato_listo.notify_all()
            self.pedidos.task_done()

    def mesero(self, id_mesero):
        while True:
            with self.cond_plato_listo:
                self.cond_plato_listo.wait()
            if self.todos_pedidos_servidos: # Verificar si ya se sirvieron todos los pedidos
                break
            with self.sem_meseros:
                self.actualizar_gui(f"Mesero {id_mesero} está sirviendo un plato.")
                time.sleep(random.uniform(0.5, 1))
            with self.cond_mesero:
                self.cond_mesero.notify() # Notificar al chef que el plato ha sido servido
```



```

def actualizar_gui(self, mensaje):
    if self.gui_ref: # Si se ha proporcionado una referencia a la GUI, usarla
        self.gui_ref.actualizar_log(mensaje)
    else:
        print(mensaje) # De lo contrario, imprimir en consola

def nuevo_pedido(self, id_pedido):
    self.actualizar_gui(f"Recibido nuevo pedido: {id_pedido}")
    self.pedidos.put(id_pedido)

def finalizar_servicio(self):
    # Asegurarse de que todos los pedidos han sido procesados
    while not self.pedidos.empty():
        time.sleep(1)
    # Colocar None en la cola para cada chef como señal de parada
    for i in range(self.num_chefs):
        self.pedidos.put(None)
    with self.cond_plato_listo:
        self.todos_pedidos_servidos = True
        self.cond_plato_listo.notify_all()
    self.actualizar_gui("Todos los pedidos han sido preparados y servidos.")

def programar_pedidos(self, total_pedidos, pedido_actual):
    if pedido_actual < total_pedidos:
        self.nuevo_pedido(pedido_actual)
        # Programar el próximo pedido para después de un tiempo aleatorio
        delay = int(random.uniform(100, 300)) # milisegundos
        self.gui_ref.master.after(delay, lambda: self.programar_pedidos(total_pedidos, pedido_actual +
1))
    else:
        # Una vez programados todos los pedidos, esperar un poco antes de detener los chefs
        self.gui_ref.master.after(1000, self.finalizar_servicio)

def iniciar_servicio(self, num_pedidos):
    # Iniciar hilos de chefs
    for i in range(self.num_chefs):
        threading.Thread(target=self.chef, args=(i,)).start()

    # Iniciar hilos de meseros
    for i in range(self.num_meseros):
        threading.Thread(target=self.mesero, args=(i,)).start()

    # Programar la llegada de pedidos
    self.programar_pedidos(num_pedidos, 0)

    # Colocar None en la cola para cada chef como señal de parada
    for i in range(self.num_chefs):
        self.pedidos.put(None)

    # Esperar a que todos los pedidos sean procesados
    self.pedidos.join()
    with self.cond_plato_listo:
        self.todos_pedidos_servidos = True
        self.cond_plato_listo.notify_all()
    self.actualizar_gui("Todos los pedidos han sido preparados y servidos.")

class RestauranteGUI:
    def __init__(self, master, restaurante):
        self.master = master
        self.restaurante = restaurante
        master.title("Simulación de Restaurante")

        # Área de log
        self.log = tk.Text(master, state='disabled', height=10)
        self.log.pack(padx=10, pady=10)

        # Botón de inicio
        self.start_button = tk.Button(master, text="Iniciar Servicio", command=self.iniciar_servicio)
        self.start_button.pack(pady=5)

        # Estado de chefs y meseros
        self.estado_chefs = tk.Label(master, text="Chefs: Esperando")
        self.estado_chefs.pack(pady=5)

        self.estado_meseros = tk.Label(master, text="Meseros: Esperando")
        self.estado_meseros.pack(pady=5)

    def actualizar_log(self, mensaje):
        # Esta función se asegura de que la actualización de la GUI se ejecute en el hilo principal.
        def log_update():
            self.log.config(state='normal')
            self.log.insert('end', mensaje + "\n")
            self.log.config(state='disabled')
            self.log.see('end')
        self.master.after(0, log_update)

    def iniciar_servicio(self):
        self.start_button.config(state='disabled')
        self.actualizar_log("Iniciando servicio...")
        # Iniciar la simulación en un hilo separado para no bloquear la GUI
        threading.Thread(target=self.simulacion).start()

    def simulacion(self):
        # Esta función se ejecuta en un hilo separado para manejar la simulación.
        # Llama a la función de inicio de servicio del restaurante.
        self.restaurante.iniciar_servicio(10)

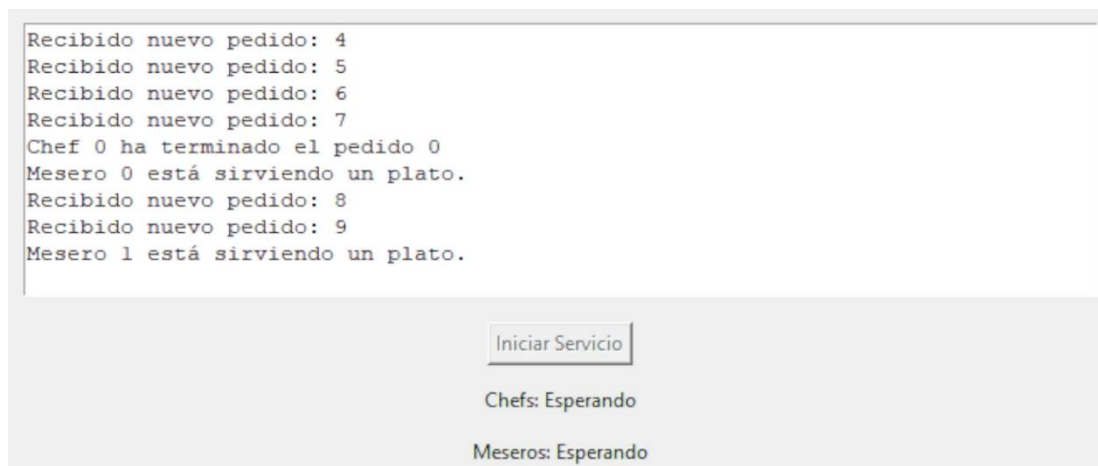
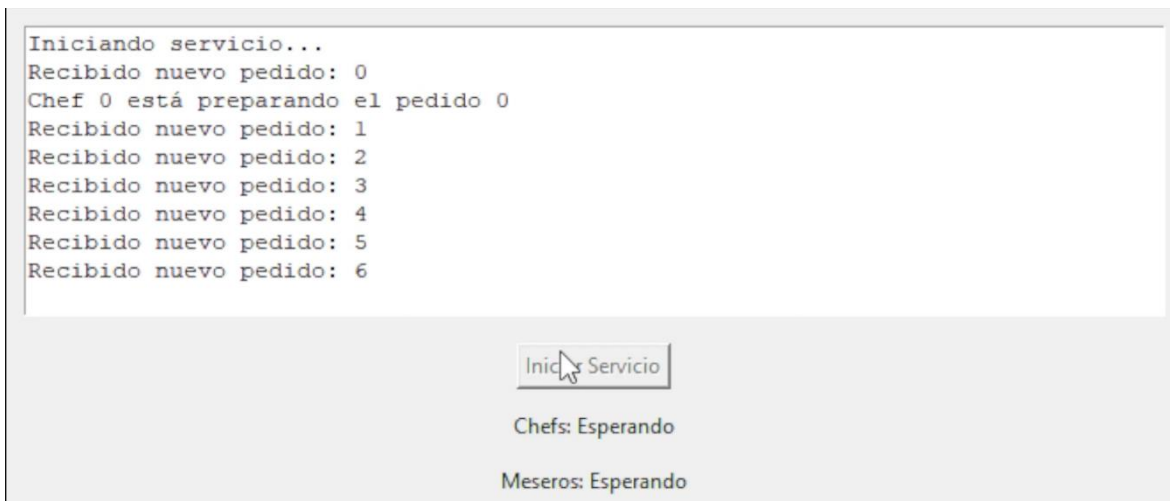
# Crear la instancia de Restaurante
restaurante = Restaurante(3, 2, None)

# Crear la instancia de Tkinter
root = tk.Tk()
app = RestauranteGUI(root, restaurante) # Pasar la instancia de Restaurante a RestauranteGUI
restaurante.gui_ref = app # Establecer la referencia de la GUI en la instancia de Restaurante

root.mainloop()

```

Pantallazo de ejecución:



A partir de aquí la interfaz dejaba de responder y se congelaba.