Simulación de un Sistema Hospitalario Automatizado

Programación Paralela, Concurrente y Asíncrona

Roberto Baeza

21 de abril de 2025

Resumen

Este informe presenta el diseño e implementación de una simulación de un sistema hospitalario automatizado utilizando paradigmas de programación **paralela**, **concurrente** y **asíncrona**. El objetivo es modelar el flujo de atención de pacientes desde su llegada hasta su alta médica, distribuyendo tareas en distintos tiempos y recursos mediante técnicas modernas de procesamiento. La simulación se implementó en Python y se evaluó su rendimiento en situaciones realistas de carga.

Introducción

La simulación de sistemas complejos como los hospitalarios permite comprender la interacción de múltiples procesos que ocurren simultáneamente. Este proyecto simula la llegada de pacientes al área de urgencias de un hospital, pasando por registro, diagnóstico, asignación de camas y alta. Se emplean distintos paradigmas para reflejar comportamientos realistas y eficientes: concurrencia para el uso compartido de recursos, paralelismo para tareas intensivas y asincronía para procesos con latencia variable.

Diagrama del Sistema

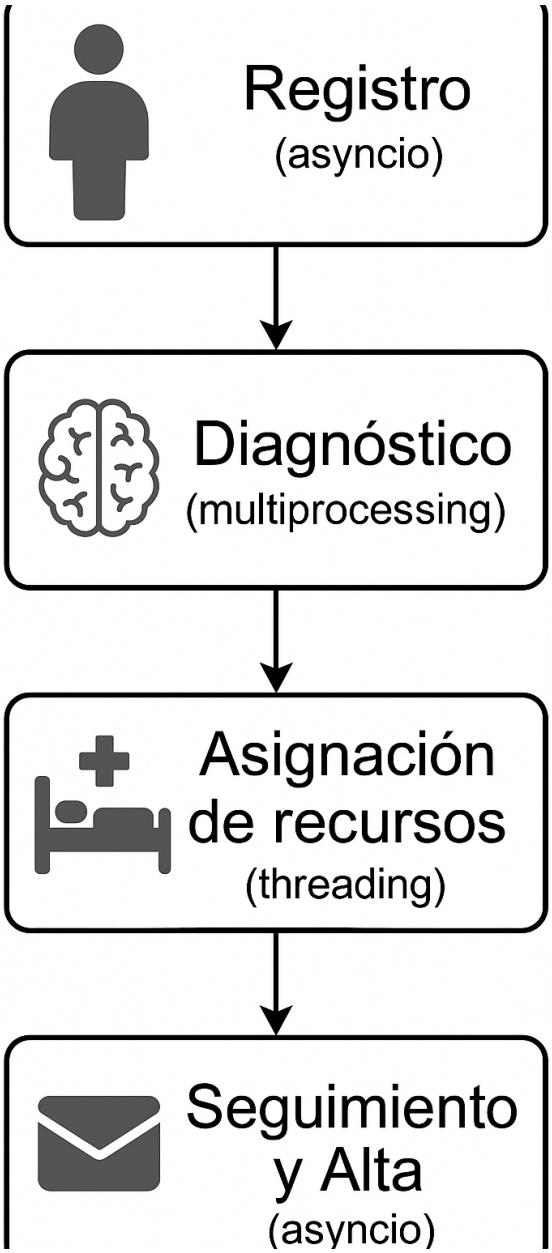

Figura 1. Flujo general del sistema hospitalario.

Diseño e Implementación

- **Registro y alta:** Usan `asyncio` para representar tareas no bloqueantes. El registro y el alta pueden ejecutarse en paralelo para varios pacientes.
- **Diagnóstico:** Modelado como una tarea pesada, ejecutada en paralelo con `multiprocessing`. Simula procesamiento intensivo.
- **Asignación de recursos:** Utiliza `threading` con semáforos que simulan camas disponibles.
- **Flujo completo:** Cada paciente es manejado en un hilo. Dentro de cada hilo se integran llamadas asíncronas, procesos paralelos y concurrencia.

Código

1. Fragmentos Clave del Código con Explicación

1.1. Importación de librerías

Listing 1: Importación de librerías

```
import asyncio              # Para tareas as ncronas
import threading            # Para concurrencia con hilos
import multiprocessing      # Para procesos en paralelo
import time
import random               # Para simular tiempos
   aleatorios
```

1.2. Funciones Asíncronas: Registro y Alta

Listing 2: Registro y alta del paciente (Asíncrono)

```
# Simula el registro de un paciente con retardo
   aleatorio
```

```
async def registro_paciente(nombre):
    print(f" Registrando a {nombre}...")
    await asyncio.sleep(random.uniform(0.5, 1.5))
    print(f" {nombre} registrado.")

# Simula el seguimiento hasta el alta del paciente
async def seguimiento_alta(nombre):
    await asyncio.sleep(random.uniform(1, 2))
    print(f" {nombre} fue dado de alta.")
```

1.3. Diagnóstico en Paralelo (Multiproceso)

Listing 3: Diagnóstico médico paralelo

```
# Simula un diagn stico m dico con procesamiento
   intensivo
def diagnostico(nombre):
    print(f" Diagn stico para {nombre} en curso...")
    time.sleep(random.uniform(2, 4))  # Tarea pesada
    print(f" Diagn stico de {nombre} finalizado.")
```

1.4. Concurrencia: Asignación de Camas

Listing 4: Asignación de camas hospitalarias (Concurrencia)

```
# Recurso compartido: solo 3 camas disponibles
camas_disponibles = threading.Semaphore(3)

# Controla el acceso concurrente a las camas usando
   sem foro
def asignar_recursos(nombre):
    print(f" {nombre} esperando cama...")
    with camas_disponibles:
        print(f" {nombre} asignado a cama.")
        time.sleep(random.uniform(1, 2))  # Simula el
           uso de la cama
        print(f" {nombre} dej  la cama disponible.")
```

1.5. Flujo Completo por Paciente

Listing 5: Flujo completo por paciente

```
# Define el ciclo completo que sigue cada paciente
def flujo_paciente(nombre):
    asyncio.run(registro_paciente(nombre))  # As ncrono

    # Diagn stico en proceso paralelo
    p = multiprocessing.Process(target=diagnostico, args
       =(nombre,))
    p.start()

    asignar_recursos(nombre)  # Concurrencia

    p.join()  # Esperar que termine el diagn stico

    asyncio.run(seguimiento_alta(nombre))  # As ncrono
```

1.6. Inicio de la Simulación

Listing 6: Simulación hospitalaria

```
if __name__ == "__main__":
    # Crear lista de pacientes
    pacientes = [f"Paciente-{i+1}" for i in range(5)]
    hilos = []

    # Crear un hilo por paciente
    for nombre in pacientes:
        hilo = threading.Thread(target=flujo_paciente,
           args=(nombre,))
        hilos.append(hilo)
        hilo.start()

    # Esperar que todos los hilos terminen
    for hilo in hilos:
        hilo.join()

    print(" Simulaci n hospitalaria finalizada.")
```

Pruebas y Resultados

La simulación se ejecutó con 5 pacientes. Los resultados muestran que:

- El registro y el alta se completan de forma rápida gracias a la asincronía.
- Solo tres pacientes están asignados a camas al mismo tiempo, como se controla con un semáforo.
- La ejecución paralela del diagnóstico permite que varios análisis se realicen simultáneamente.
- El sistema maneja correctamente múltiples hilos y procesos sin errores.

1.7. Resultados obtenidos

A continuación se muestra un ejemplo de la salida obtenida al ejecutar el código de simulación hospitalaria:

Listing 7: Salida de la simulación

```
Registrando a Paciente-1...
Registrando a Paciente-2...
Registrando a Paciente-3...
Registrando a Paciente-4...
Registrando a Paciente-5...
Paciente-2 registrado.
Paciente-2 esperando cama...
Paciente-2 asignado a cama.
Diagn stico para Paciente-2 en curso...
Paciente-3 registrado.
Paciente-3 esperando cama...
Paciente-3 asignado a cama.
Diagn stico para Paciente-3 en curso...
Paciente-1 registrado.
Paciente-1 esperando cama...
Paciente-1 asignado a cama.
Diagn stico para Paciente-1 en curso...
Paciente-4 registrado.
Paciente-4 esperando cama...
Paciente-5 registrado.
Diagn stico para Paciente-4 en curso...
```

```
Paciente-5 esperando cama...
Diagn stico para Paciente-5 en curso...
Paciente-2 dej  la cama disponible.
Paciente-4 asignado a cama.
Paciente-3 dej  la cama disponible.
Paciente-5 asignado a cama.
Paciente-1 dej  la cama disponible.
Paciente-4 dej  la cama disponible.
Diagn stico de Paciente-2 finalizado.
Paciente-5 dej  la cama disponible.
Diagn stico de Paciente-3 finalizado.
Diagn stico de Paciente-1 finalizado.
Diagn stico de Paciente-5 finalizado.
Paciente-2 fue dado de alta.
Diagn stico de Paciente-4 finalizado.
Paciente-1 fue dado de alta.
Paciente-3 fue dado de alta.
Paciente-5 fue dado de alta.
Paciente-4 fue dado de alta.
Simulaci n hospitalaria finalizada.
```

Conclusiones

El proyecto permitió aplicar y diferenciar claramente los paradigmas de programación paralela, concurrente y asíncrona. El uso de Python facilitó la combinación de `threading`, `multiprocessing` y `asyncio`. La simulación se comportó de manera estable y realista. Como mejora futura, se podría agregar una interfaz visual o mediciones más detalladas de rendimiento.

Referencias

- Python Docs - asyncio: `https://docs.python.org/3/library/asyncio.html`
- Python Docs - multiprocessing: `https://docs.python.org/3/library/multiprocessing.html`

- Python Docs - threading: `https://docs.python.org/3/library/threading.html`
- OpenAI ChatGPT (asistencia para estructura y redacción)