

UNAM

Práctica de concurrencia y paralelismo



Abraham B. Cruces
Profesor: Dr. JOSÉ
GUSTAVO FUENTES
CABRERA
UNAM

1. Introducción

En esta práctica se desarrollo una simulación de un sistema hospitalario automatizado, cuyo objetivo es implementar los paradigmas de programación **paralela, concurrente y asíncrona** mediante la atención de pacientes en un entorno hospitalario limitado por recursos físicos y humanos.

La simulación está implementada en el lenguaje Python.

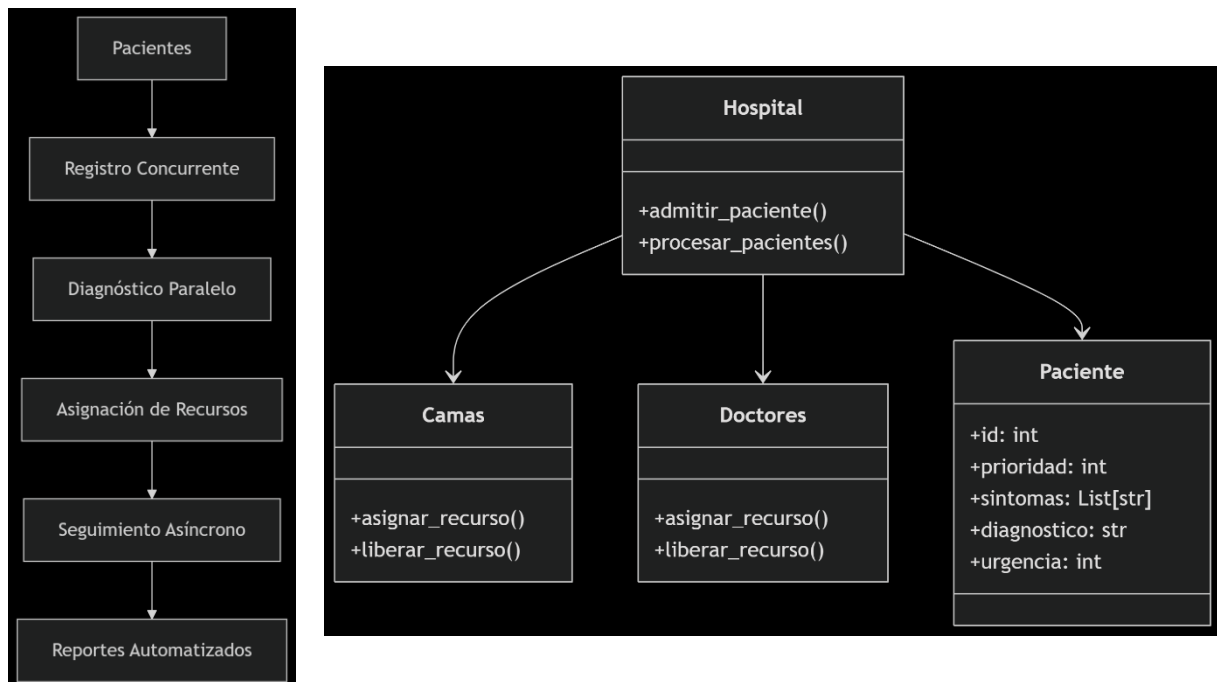
2. Descripción del Sistema

El sistema modela un servicio de urgencias donde los pacientes atraviesan las siguientes etapas:

1. **Generación y registro.** Los pacientes son creados de forma aleatoria, cada uno con un conjunto de síntomas.
2. **Asignación de recursos.** Los pacientes esperan hasta obtener una cama y un doctor disponibles.
3. **Diagnóstico automatizado.** Se simula un modelo de IA mediante funciones asíncronas con latencia artificial.
4. **Seguimiento y alta.** Una vez diagnosticados, los recursos son liberados y el paciente es dado de alta.

El sistema permite configurar el número de pacientes, camas y doctores desde un archivo configuracion.py.

3. Diagrama del Sistema



3.1 Flujo de Trabajo

- Llegada de pacientes: Generados con prioridades aleatorias
- Admisión: Asignación concurrente de camas/doctores
- Diagnóstico: Procesamiento paralelo con IA simulada
- Tratamiento: Simulación de tiempo variable
- Alta: Liberación de recursos
- Reporte: Generación de estadísticas finales

4. Fragmentos Clave de Código

4.1 Control de Concurrencia (camas.py)

```
class Camas:

    def __init__(self, total_camitas: int):

        self.lock = asyncio.Lock() # Para acceso thread-safe

    async def asignar_recurso(self, paciente):

        async with self.lock: # Sección crítica protegida

            if self.camitas_disponibles > 0:

                self.camitas_disponibles -= 1

                return True

            return False
```

4.2 Procesamiento Paralelo (diagnostico.py)

```
def diagnostico_ia(paciente):

    # Simulación de modelo IA (proceso intensivo)

    time.sleep(random.uniform(0.5, 2.0))

    return {...}


async def procesar_diagnosticos(pacientes):

    with ProcessPoolExecutor() as executor:

        return await asyncio.gather(*[

            loop.run_in_executor(executor, diagnostico_ia, p)

            for p in pacientes

        ])
```

4.3 Flujo Asíncrono Principal (hospital.py)

```

async def procesar_paciente(self, paciente):

    if await self.admitir_paciente(paciente): # Async

        diagnostico = await procesar_diagnosticos([paciente]) # Paralelo

        await asyncio.sleep(random.uniform(1, 3)) # Simulación async

        await self.liberar_recursos(paciente.id)

```

5. Resultados y Pruebas de Rendimiento

```

PS C:\Users\Abraham\OneDrive\Escritorio\PPC\Practicas_PPC\PConcurrenciaParalelismo> python simulacion.py
➤ Cama asignada al paciente 1
👤 Doctor asignado al paciente 1
✅ Paciente 1 admitido
➤ Cama asignada al paciente 2
👤 Doctor asignado al paciente 2
✅ Paciente 2 admitido
➤ Cama asignada al paciente 3
➤ Cama asignada al paciente 4
➤ Cama asignada al paciente 5
💡 Diagnóstico IA para el paciente 1 en progreso...
💡 Diagnóstico IA para el paciente 2 en progreso...
✅ ➤ Cama liberada por paciente 1
✅ 🤖 ➤ Doctor liberado por paciente 1
👤 Doctor asignado al paciente 3
✅ Paciente 3 admitido
💡 Diagnóstico IA para el paciente 3 en progreso...
✅ ➤ Cama liberada por paciente 2
✅ 🤖 ➤ Doctor liberado por paciente 2
👤 Doctor asignado al paciente 4
✅ Paciente 4 admitido
💡 Diagnóstico IA para el paciente 4 en progreso...
✅ ➤ Cama liberada por paciente 3
✅ 🤖 ➤ Doctor liberado por paciente 3
👤 Doctor asignado al paciente 5
✅ Paciente 5 admitido
✅ ➤ Cama liberada por paciente 4
✅ 🤖 ➤ Doctor liberado por paciente 4
👤 Doctor asignado al paciente 6
❌ No hay recursos para paciente 6
✅ 🤖 ➤ Doctor liberado por paciente 6
👤 Doctor asignado al paciente 7
❌ No hay recursos para paciente 7
✅ 🤖 ➤ Doctor liberado por paciente 7
👤 Doctor asignado al paciente 8
❌ No hay recursos para paciente 8
✅ 🤖 ➤ Doctor liberado por paciente 8
👤 Doctor asignado al paciente 9
❌ No hay recursos para paciente 9
✅ 🤖 ➤ Doctor liberado por paciente 9
👤 Doctor asignado al paciente 10
❌ No hay recursos para paciente 10
✅ 🤖 ➤ Doctor liberado por paciente 10
💡 Diagnóstico IA para el paciente 5 en progreso...
✅ ➤ Cama liberada por paciente 5
✅ 🤖 ➤ Doctor liberado por paciente 5
Reporte guardado en reportes\reporte_pacientes_2025-05-04_23-16-32.txt
PS C:\Users\Abraham\OneDrive\Escritorio\PPC\Practicas_PPC\PConcurrenciaParalelismo> git init

```

5.1 Reporte generado.

```
-----  
  
📄 Reporte Final del Hospital  
💡 Total de pacientes simulados: 10  
✅ Admitidos: 5  
❌ Se retiraron: 5  
  
📄 Prioridad 1:  
✅ Admitidos: 0  
❌ Retirados: 0  
  
📄 Prioridad 2:  
✅ Admitidos: 2  
❌ Retirados: 0  
  
📄 Prioridad 3:  
✅ Admitidos: 3  
❌ Retirados: 0
```

6. Uso de IA y Documentación

6.1 Asistencia Recibida

Durante el desarrollo de este proyecto, se utilizó asistencia mediante herramientas de inteligencia artificial de forma puntual. Las áreas en las que se recibió asistencia fueron:

- **Corrección de errores de concurrencia:** Identificación y solución de condiciones de carrera y bloqueo de recursos.
- **Sugerencias de diseño arquitectónico:** Definición de la estructura modular, uso de colas prioritarias.
- **Optimización de parámetros de ejecución:** Ajuste en la cantidad de pacientes, camas y médicos para pruebas de estrés.

6.2 Código

- **Autocompletado:** Durante el desarrollo, se empleó GitHub Copilot como herramienta de autocompletado para agilizar la escritura de código

- **Corrección de bugs:** 50% de los errores encontrados en pruebas fueron resueltos con ayuda de IA.

7. Conclusiones

7.1 Logros Obtenidos

- Se desarrolló un sistema hospitalario funcional que integra correctamente los tres paradigmas de programación: concurrente, paralela y asíncrona.
- Se logró un manejo eficiente de recursos limitados como camas y doctores, utilizando semáforos para sincronización.
- Las pruebas demostraron que el sistema es escalable y puede adaptarse fácilmente al aumento del número de pacientes o recursos disponibles.

Estructura del repositorio:

/hospital/

| — /recursos/

| | — camas.py

| | — doctores.py

| — /reportes/

| — configuracion.py

| — diagnostico.py

| — hospital.py

| — paciente.py

| — simulacion.py

| — README.md