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.

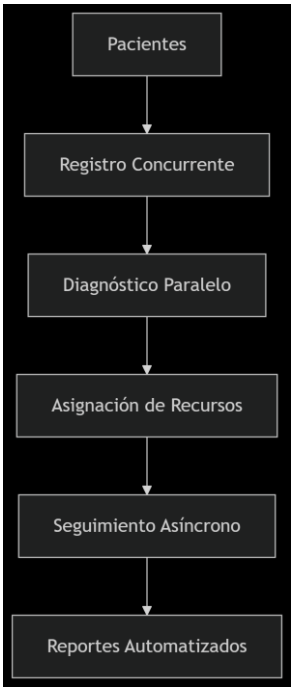
2. Descripción del Sistema

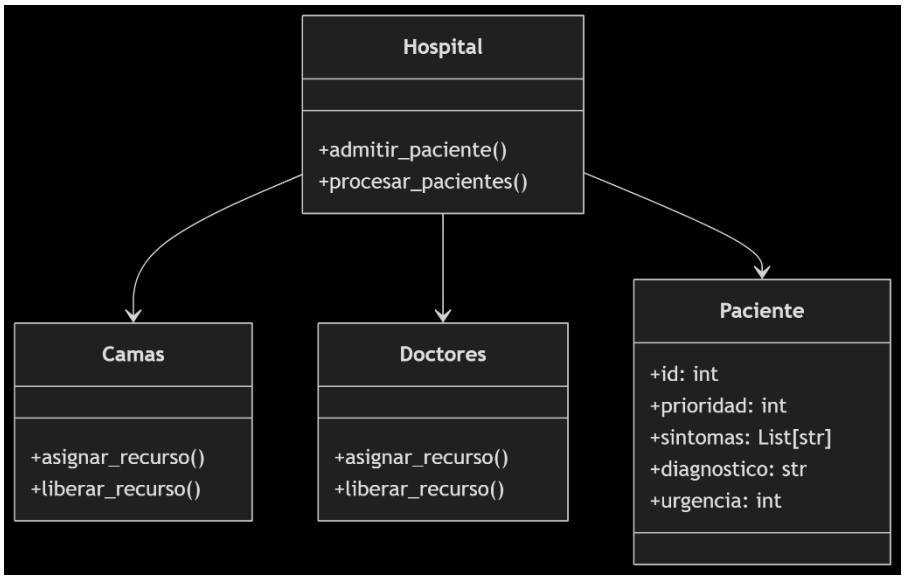
El sistema modela un servicio de urgencias donde los pacientes atraviesan las siguientes etapas:

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
```

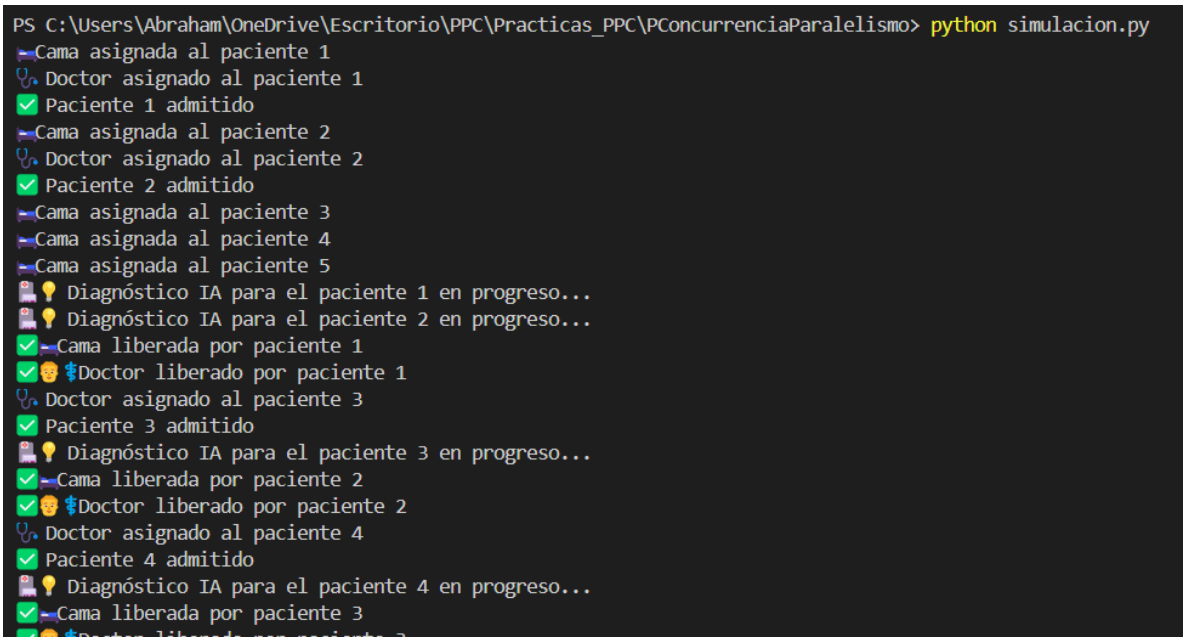
4.2 Procesamiento Paralelo (diagnostico.py)

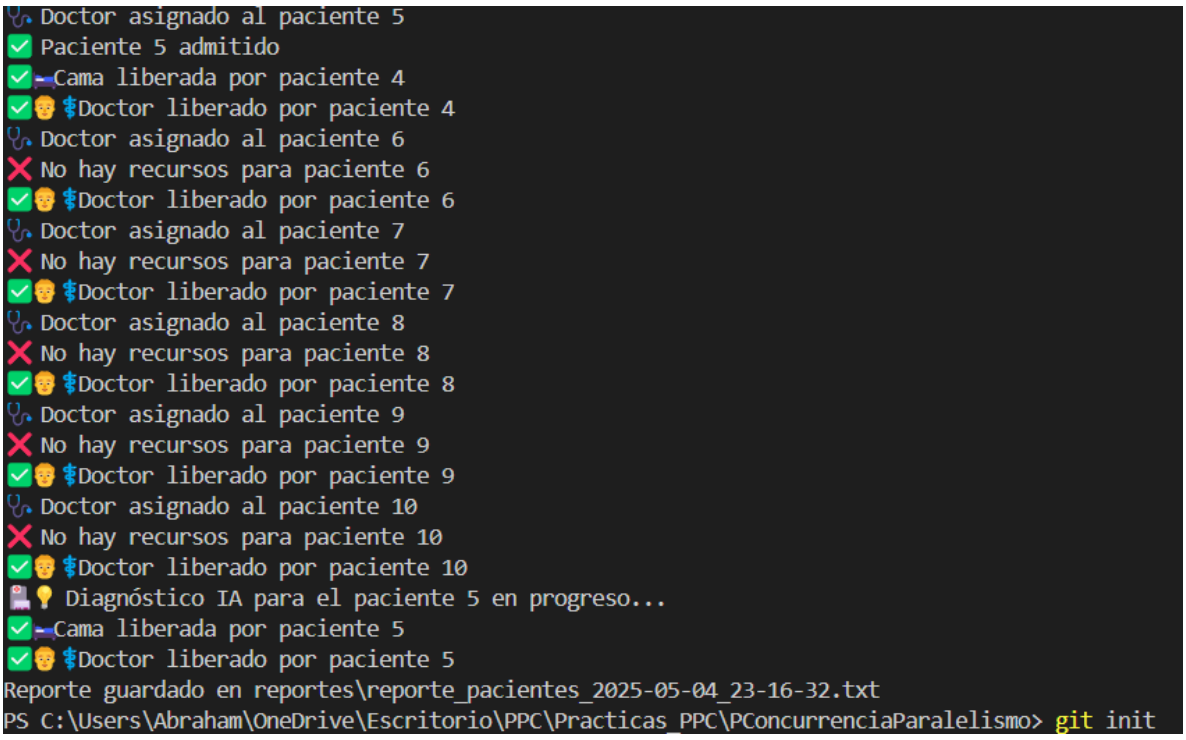
```
def diagnostico_ia(paciente):
  # Simulación de modelo IA (proceso intensivo)
  time.sleep(random.uniform(0.5, 2.0))
  return {...}


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)

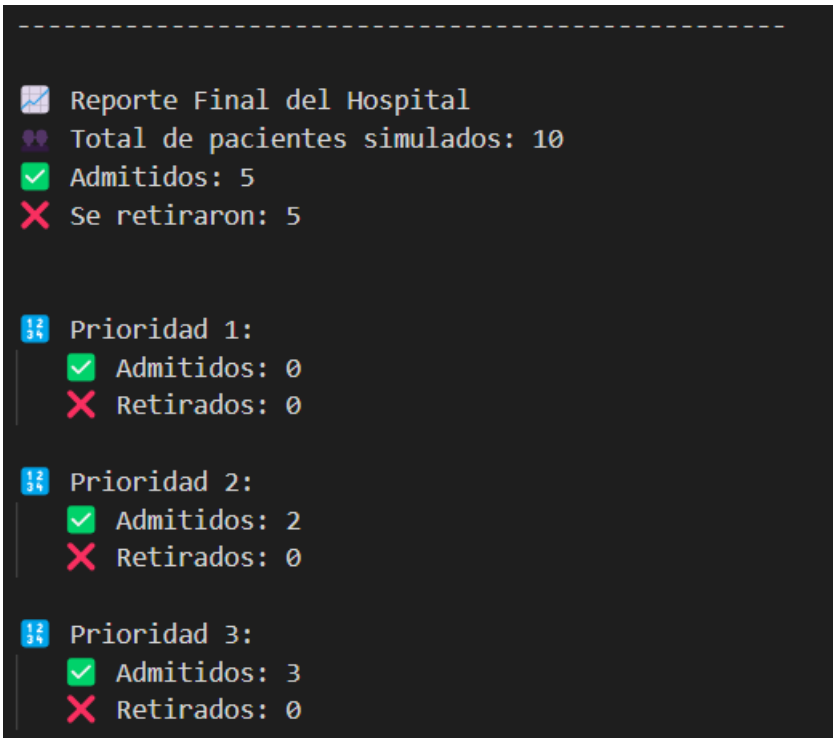
```
async def procesar_paciente(self, paciente):
    if await self.admitir_paciente(paciente):  # Async
        diagnostico = await procesar_diagnosticos([paciente])  # Paralelo
        await asyncio.sleep(random.uniform(1, 3))  # Simulación async
        await self.liberar_recursos(paciente.id)
```

5. Resultados y Pruebas de Rendimiento





5.1 Reporte generado.



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 Codigo

- **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