Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2024

Proyecto 3 Digital Intellifne

202307272 – Bryan Alejandro Anona Paredes

Resumen

El proyecto descrito en este ensayo se centra en el desarrollo de un sistema de gestión de máquinas y productos utilizando Python y Flask como marco de trabajo para la interfaz web. El sistema permite el procesamiento de archivos XML para cargar información de máquinas y productos, y posteriormente ejecutar instrucciones de ensamblaje mediante el uso de listas enlazadas y la ejecución paralela de hilos. El uso de listas enlazadas personalizadas para almacenar los datos y el manejo de un ensamblaje concurrente son aspectos clave de esta implementación, lo que lo hace un sistema eficiente para la gestión de procesos industriales automatizados. A lo largo de este trabajo, se detallan los componentes principales del código y se discuten las decisiones de diseño que permitieron optimizar el rendimiento del sistema.

Palabras clave

Flask, Python, Listas Enlazadas, Ensamblaje, Multihilo

Resume

The project described in this essay focuses on the development of a machine and product management system using Python and Flask as the framework for the web interface. The system allows for the processing of XML files to load information about machines and products, and subsequently execute assembly instructions through the use of linked lists and parallel thread execution. The use of custom linked lists to store data and the handling of concurrent assembly are key aspects of this implementation, making it an efficient system for managing automated industrial processes. Throughout this work, the main components of the code are detailed, and the design decisions that allowed for the optimization of system performance are discussed.

Keywords

Flask, Python, Linked Lists, Assembly, Multithreading

Introducción

Los avances en la automatización industrial han hecho imprescindible la creación de sistemas eficientes para gestionar máquinas y productos en el proceso de montaje.

Este artículo describe el desarrollo de un sistema basado en Python y Flask que utiliza estructuras de datos personalizadas, como listas vinculadas y concurrencia de subprocesos, para controlar el ensamblaje en diferentes líneas de producción.

Discutiremos los principales elementos de este sistema y las soluciones implementadas para mejorar su rendimiento.

Desarrollo del tema

1. Descripción del backend en Flask

El sistema está construido utilizando Flask, un microframework ligero de Python que facilita el desarrollo de aplicaciones web. Flask se destaca por su simplicidad y flexibilidad, lo que lo convierte en una excelente opción para proyectos donde se requiere una interacción rápida entre el usuario y el sistema, como en este caso. El archivo frontend.py gestiona la lógica del servidor web, permitiendo que los usuarios carguen archivos XML y reciban reportes personalizados de las máquinas y productos almacenados en el sistema.

El punto de entrada del sistema es la función analizarDatosXML, que recibe un archivo XML cargado por el usuario a través de un formulario web. Una vez recibido, el archivo es procesado usando minidom, una herramienta de Python para manipular XML. Esta función extrae la información relevante de cada máquina y producto, como el nombre, la cantidad de líneas de producción, la cantidad de componentes y las instrucciones de ensamblaje para cada producto. Cada uno de estos datos se almacena en una lista enlazada personalizada, evitando el uso de listas nativas de Python.

Este enfoque modular facilita la expansión del sistema. Si, por ejemplo, se requiere agregar nuevas características, como reportes adicionales o diferentes tipos de procesos de ensamblaje, estas se pueden integrar sin afectar el flujo existente. Además, Flask proporciona una capa de abstracción entre el usuario y el sistema, lo que significa que cualquier actualización en la lógica interna del sistema no afectará directamente a la interfaz de usuario.

2. Estructura de datos personalizada: Listas enlazadas

Uno de los aspectos más interesantes de este proyecto es el uso de listas enlazadas para manejar tanto las máquinas como los productos. Las listas enlazadas son estructuras de datos eficientes que permiten agregar, eliminar y acceder a elementos de manera dinámica sin tener que redefinir el tamaño de la estructura.

La clase ListaMaquinas gestiona una lista enlazada de objetos Maquina. Esta estructura permite agregar máquinas a medida que se procesan en el archivo XML, manteniendo un enlace entre cada máquina y sus sucesoras. La ventaja principal de esta implementación es que permite un manejo eficiente de memoria y evita el uso de estructuras más pesadas, como diccionarios o listas nativas de Python, que pueden ser menos eficientes en términos de acceso y modificación.

De manera similar, la clase ListaProductos sigue el mismo principio para almacenar productos. Cada Producto contiene un conjunto de instrucciones que describen cómo debe ser ensamblado, y estas instrucciones se almacenan en una cola enlazada, gestionada por la clase ColaInstrucciones. Este diseño no solo permite una gestión eficiente de los datos, sino que también es escalable: se pueden agregar nuevos productos y máquinas sin necesidad de modificar la estructura de las listas.

Además, al usar estas estructuras enlazadas, se facilita la iteración sobre los elementos, como se muestra en las funciones mostrar() de las clases Maquina y ListaMaquinas. Esto permite que la información de cada máquina y sus productos se

despliegue de forma organizada, ofreciendo una visión clara de los datos procesados.

3. Manejo de colas e instrucciones de ensamblaje

El ensamblaje de productos se realiza a través de instrucciones almacenadas en una cola de instrucciones. Esta cola está implementada como una lista enlazada donde cada instrucción se encola y desencola a medida que se ejecutan. La clase ColaInstrucciones gestiona estas colas, que permiten procesar las instrucciones de ensamblaje en el orden en que se recibieron.

Cada producto tiene su propia cola de instrucciones, lo que asegura que el ensamblaje siga el flujo correcto. El uso de colas es especialmente útil en entornos de producción industrial, donde es crucial que las tareas se realicen en un orden determinado. Por ejemplo, un brazo robótico no puede ensamblar una pieza si la anterior no ha sido ensamblada correctamente. En este proyecto, las colas aseguran que las instrucciones se procesen de forma secuencial, simulando este tipo de comportamiento.

Un aspecto clave de esta implementación es que las colas de instrucciones están completamente desacopladas de la lógica de la máquina. Esto significa que las instrucciones para cada producto pueden ser gestionadas de manera independiente, permitiendo que múltiples productos se ensamblen simultáneamente en diferentes líneas de producción, cada una con su propia cola de instrucciones.

Además, la estructura de cola facilita la integración de nuevas funcionalidades, como la posibilidad de pausar o reordenar instrucciones si fuera necesario. Esta flexibilidad en la gestión de las tareas es esencial para sistemas de ensamblaje industrial, donde es común que los procesos deban ajustarse dinámicamente en tiempo real.

4. Ejemplo de ensamblaje multihilo con la clase Robot

El ensamblaje de los productos es simulado mediante el uso de hilos (threads). Cada brazo robótico que opera en una línea de producción es representado por un hilo independiente, lo que permite que múltiples líneas operen en paralelo. Este enfoque multihilo es particularmente útil en sistemas donde se deben realizar múltiples tareas simultáneamente para optimizar el tiempo de producción.

La clase Robot incluye una función llamada mover_brazo, que simula el movimiento de un brazo robótico desde una posición inicial hasta una posición objetivo en una línea de producción. Cada movimiento es ejecutado por un hilo diferente, lo que significa que cada línea de producción puede operar de manera independiente, sin esperar a que otras líneas terminen sus tareas.

Un aspecto importante del diseño es el uso de un "lock" para controlar el ensamblaje. El "lock" garantiza que solo un brazo robótico pueda ensamblar en un componente determinado en un momento dado. Esto es crucial en escenarios de producción donde múltiples brazos podrían intentar ensamblar piezas en la misma línea al mismo tiempo, lo que podría causar errores.

La función ejecutar_maquina gestiona el ensamblaje de un producto. Esta función itera sobre la cola de instrucciones de un producto y asigna un hilo para cada movimiento del brazo robótico. Al final del proceso, se espera que todos los hilos hayan completado sus tareas antes de que el ensamblaje sea considerado finalizado. Este enfoque asegura que el sistema sea eficiente y permita la ejecución de múltiples productos en paralelo, maximizando el uso del tiempo y los recursos disponibles.

5. Interacción entre Máquina y Producto

La interacción entre las clases Maquina y Producto es un ejemplo de un diseño modular eficiente. Cada Maquina contiene una lista de productos que se ensamblan en sus líneas de producción, y cada Producto tiene una cola de instrucciones que describe cómo debe ser ensamblado. Este diseño permite que las máquinas y productos sean gestionados de manera independiente, mientras que la cola de instrucciones proporciona la flexibilidad necesaria para controlar el proceso de ensamblaje de manera precisa.

La relación entre ambas clases se gestiona mediante listas enlazadas, lo que permite agregar productos y máquinas de forma dinámica. Además, este enfoque

facilita la extensión del sistema; por ejemplo, se podrían agregar nuevas características a los productos o ajustar las líneas de producción de una máquina sin necesidad de modificar el funcionamiento general del sistema.
En resumen, la interacción entre máquina y producto se basa en un diseño modular que maximiza la eficiencia y permite una fácil escalabilidad. El uso de listas enlazadas y colas personalizadas asegura que el sistema pueda manejar de manera efectiva una gran cantidad de datos y operaciones simultáneas, lo que lo convierte en una solución ideal para entornos de producción automatizada.

Conclusiones

Este proyecto muestra cómo combinar Flask y Python con estructuras de datos personalizadas puede ser una solución útil para la gestión de procesos de ensamblaje automatizado. La ejecución paralela mediante hilos y las listas enlazadas optimizan el rendimiento del sistema y facilitan la escalabilidad. Este método ofrece un marco flexible para integrar nuevas funcionalidades en futuros desarrollos, lo que lo convierte en una opción viable para proyectos de automatización industrial de mayor envergadura.

Anexos:

1. Método para Cargar Instrucciones desde un Archivo

Este método permite cargar instrucciones de ensamblaje desde un archivo de texto, lo que puede ser útil para alimentar la cola de instrucciones de un producto.

python
Copiar código

```
def cargar_instrucciones_desde_archivo(self, archivo_instrucciones):
    with open(archivo_instrucciones, 'r') as file:
        for linea in file:
            instruccion = linea.strip()
            self.cola_instrucciones.encolar(instruccion)
            print(f"Instrucción encolada desde archivo: {instruccion}")
```

2. Método para Mostrar Detalles de la Máquina

Este método adicional puede incluirse en la clase Maquina para mostrar detalles más específicos sobre la máquina y sus productos.

python
Copiar código

```
def mostrar_detalles(self):
    print(f"Detalles de la Máquina: {self.nombre}")
    print(f"Líneas de Producción: {self.n}, Componentes: {self.m}, Tiempo de Ensamblaje: {self.tiempoEnsamblaje}")
    print("Productos asociados:")
    self.productos.mostrar()  # Llama al método mostrar de ListaProductos
```

3. Método para Verificar la Disponibilidad de Productos

Este método permite verificar si hay productos disponibles en la máquina antes de ejecutar el ensamblaje.

python
Copiar código

```
def tiene_productos_disponibles(self):
    return self.productos.cabeza is not None
```

4. Método para Reiniciar la Cola de Instrucciones

Un método que permite reiniciar la cola de instrucciones para un producto, útil si se desea repetir el proceso de ensamblaje.

python
Copiar código

```
def reiniciar_instrucciones(self):
    self.cola_instrucciones = ColaInstrucciones()
    print(f"La cola de instrucciones para {self.nombre} ha sido reiniciada.")
```

5. Método de Ejemplo para Manejar Errores

Este método maneja errores durante el proceso de ensamblaje, proporcionando un mensaje adecuado al usuario.

python
Copiar código

```
def manejar_error(self, error):
```

```
    print(f"Error en el proceso de ensamblaje: {error}. Se recomienda verificar las instrucciones y los componentes.")
```

6. Método para Obtener el Estado de la Máquina

Un método que devuelve un estado que indica si la máquina está ocupada o lista para ensamblar.

```
python
Copiar código
def obtener_estado(self):
    if self.productos.tiene_productos_disponibles():
        return "Lista para ensamblar"
    else:
        return "No hay productos disponibles"
```

7. Implementación del Método __str__ en Producto

Una implementación más detallada del método __str__ que podría incluir más información sobre el producto.

```
python
Copiar código
def __str__(self):
    return f"Nombre del Producto: {self.nombre}, Elaboración: {self.elaboracion}, Instrucciones Pendientes: {self.cola_instrucciones.esta_vacia()}"
```

Descripción de los Fragmentos:

- **Carga de Instrucciones**: Permite cargar instrucciones desde un archivo, facilitando la inicialización del sistema.
- **Mostrar Detalles de la Máquina**: Proporciona una visualización más rica de la máquina y sus productos.
- **Verificación de Disponibilidad**: Ayuda a prevenir errores al intentar ensamblar un producto sin disponibilidad.
- **Reinicio de Instrucciones**: Permite comenzar de nuevo el proceso de ensamblaje si es necesario.
- **Manejo de Errores**: Mejora la robustez del sistema al manejar fallos durante el ensamblaje.
- **Estado de la Máquina**: Ofrece información útil sobre la disponibilidad de la máquina.
- **Método __str__ en Producto**: Facilita la depuración y visualización de información relevante de los productos.