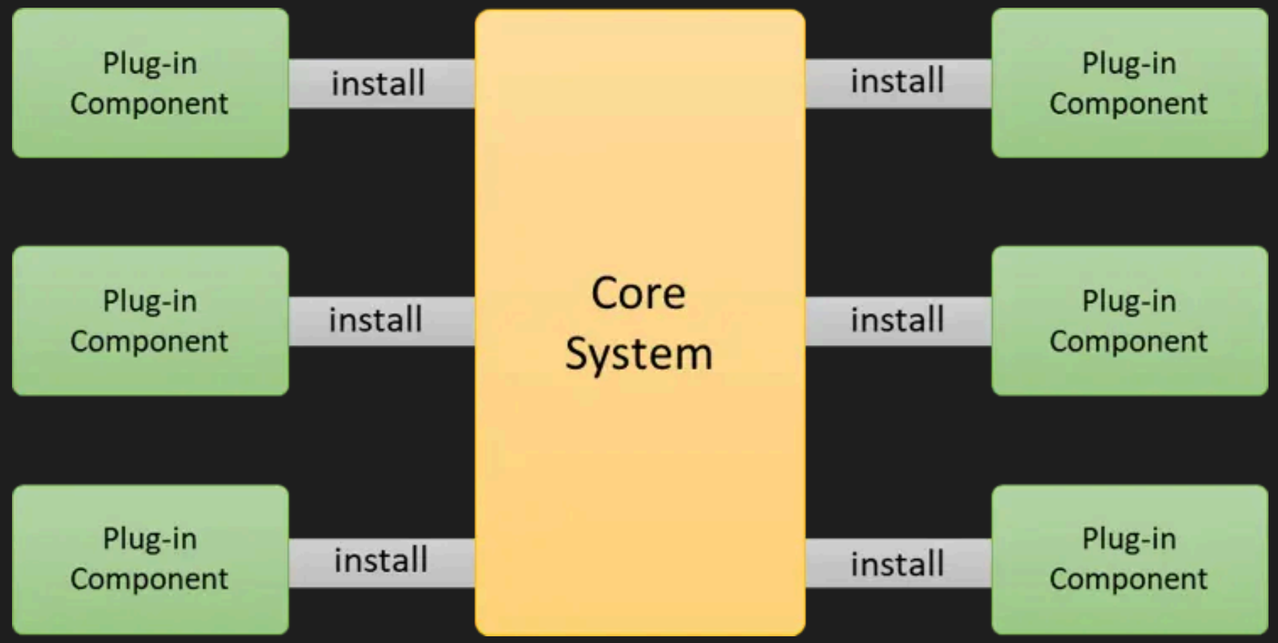
MICROKERNEL

Antonio Fernández, Josué Mena, Marco Rodríguez y Max Latysh



Descripción

La arquitectura microkernel se basa en dos componentes básicos, un sistema core y sus módulos plugin, el sistema Core contiene los elementos mínimos para hacer que la aplicación funcione y cumpla el propósito para el cual fue diseñada, los módulos o plugins son componentes periféricos que se añaden o instalan al componente Core para extender su funcionalidad.

Cuándo y dónde usar?

Básicamente es de mucha utilidad en aplicaciones que son bastante grandes y se busca extender la funcionalidad que tiene en un futuro, pero no modificarla como tal. Esto basado en el principio Open-Closed. Se usa por medio de API's que le indican a sus correspondientes plug-ins lo que se debe hacer. El plug-in implementa la API y se extiende la funcionalidad del sistema.

Ejemplos Generales

Cualquier sistema que utiliza “plugins” “dinámicos” se puede interpretar como un Microkernel. Por ejemplo, VSCode sigue al patrón porque inicialmente es un editor relativamente simple que podemos sobreextender cuando añadimos extensiones “dinámicas”.

Escenario Específico

Pseudo-Aplicación de pseudo-C++ para manejar, desde el CLI, plugins para un sistema de manejo de peces. Todos sus amigos aman pescar y quieren manejar un registro de los pescados que han pescado. Sin embargo, cada uno tiene sus propias necesidades (uno quiere solamente contar Salmón, otro quiere poder acceder al registro desde su barco, etc).

```
void añadir_fish_plugin(char *camino_al_plugin) {
 plugins.push_back(load_plugin(camino_al_plugin));
}
...

unsigned char shared_memory[MAX_MEM];
queue<message> inter_process_comunication_queue;

...
void *fish_mkernel() {
 while (1) {
  manage_memory();
  process_messages();
 }
 return 0;
}
```

```
int main() {
 ...
 pthread_create(&tid, NULL, fish_mkernel, &tid);
 ...
 while (1) {
  switch(entrada_usuario()) {
   ...
   case AGREGAR_PLUGIN:
    añadir_fish_plugin(entrada_usuario());
    break;
   ...
  }
 }
 ...
 return 0;
}
```

Implicaciones

Implementación más compleja que otros patrones

Se requiere un diseño cuidadoso para garantizar cohesion e interoperabilidad

Separar el Core y plugins implica diseñar correctamente sus interfaces

Costos

Inicialmente más tiempo y recursos por su complejidad de diseño

Mantener un ecosistema de plugins es costoso cada vez que se actualice el Core

Porque puede requerir ajustes en los plugins existentes para mantener compatibilidad

Beneficios

Gran capacidad para extender y personalizar el sistema

Flexibilidad al cambio gracias a sus plugins. La modularidad facilita su mantenimiento.

Plugins pueden ser reutilizados en diferentes proyectos.

Interoperabilidad: Estándares, protocolos, tecnologías y mecanismos que permiten que los datos fluyan entre diversos sistemas con mínima intervención humana

Mucha gracias por su atención!