Conceptos relacionados con la programación multiproceso

Programación asíncrona

- **Asynchronous I/O (Async I/O)**: Permite que una aplicación realice operaciones de entrada/salida sin bloquear el flujo de ejecución principal. Es común en aplicaciones que realizan muchas operaciones de red o de archivo.
- **Promises** y **Futures**: Mecanismos que representan el resultado de una operación asíncrona que puede completarse en el futuro. Permiten manejar resultados de forma no bloqueante.

Modelo actor

- **Actor Model**: Un modelo de concurrencia en el que los «actores» son unidades independientes de procesamiento que se comunican mediante el envío de mensajes. Este modelo ayuda a gestionar la concurrencia y evita problemas de sincronización y bloqueo.

Desempeño y escalabilidad

- **Load Balancing**: Distribuir la carga de trabajo entre varios recursos (por ejemplo, servidores o procesos) para mejorar la eficiencia y evitar cuellos de botella.
- **Horizontal vs Vertical Scaling**: Escalar horizontalmente implica añadir más máquinas o procesos, mientras que escalar verticalmente implica aumentar la capacidad de una sola máquina o proceso.

Teoría de colas

- **Queue Theory**: El estudio matemático de cómo las solicitudes de servicio se acumulan y se procesan en sistemas con recursos limitados. Es fundamental para entender el comportamiento y el rendimiento en sistemas multiproceso.

Memoria distribuida

- **Distributed Shared Memory (DSM)**: Un enfoque que permite a diferentes nodos en una red compartir una memoria virtual común, a pesar de que físicamente se encuentra en diferentes ubicaciones.

Procesamiento de eventos

- **Event-Driven Programming**: Un paradigma en el que el flujo del programa está determinado por eventos, como la llegada de datos o la interacción del usuario. Es útil para aplicaciones que requieren alta reactividad.

Control de concurrencia

- **Transactional Memory**: Un enfoque para manejar la concurrencia en sistemas multiproceso que utiliza transacciones para asegurar la consistencia de la memoria compartida sin necesidad de bloqueos.

Middleware

- **Middleware**: Software que actúa como intermediario entre diferentes aplicaciones o servicios, facilitando la comunicación y la gestión de datos en sistemas distribuidos.

Concurrency control

- **Optimistic and Pessimistic Concurrency Control**: Estrategias para manejar el acceso concurrente a recursos compartidos. La optimista asume que los conflictos son raros y maneja los conflictos cuando ocurren, mientras que la pesimista previene los conflictos mediante bloqueos.

Contenedores y orquestación

- **Containers (e.g., Docker)**: Permiten empaquetar aplicaciones y sus dependencias en un entorno aislado que puede ser ejecutado en cualquier sistema compatible.
- **Orchestration (e.g., Kubernetes)**: Herramientas para gestionar y coordinar la ejecución de contenedores en un clúster de máquinas.

Microservicios

- **Microservices architecture**: Un enfoque en el que una aplicación se divide en servicios independientes, cada uno con su propio proceso, que se comunican entre sí mediante API o mensajería.

Benchmarking y profiling

- **Benchmarking**: El proceso de medir el desempeño de un sistema o aplicación para compararlo con estándares o sistemas similares.
- **Profiling**: La técnica de analizar el comportamiento de un programa, especialmente el uso de recursos y tiempos de ejecución, para identificar cuellos de botella y áreas de mejora.