Programacion Paralela y Distribuida

Aspectos de la Programación Paralela

Practica N°2

Martin Farres - 13495

Ejercicio 1:

Modelo De Comunicacion - (Agrego Ejemplos)

Ejemplos:

- Comunicacion local: en la implementación de un algoritmo de quick sort, cada sublista ordenada puede ser intercambiada solo entre los hilos que están procesando esos datos, sin involucrar a otras partes del sistema.
- Comunicacion Global: en un modelo de MapReduce donde los datos generados por la fase "Map" en varios nodos se envían a nodos para la fase "Reduce", que puede involucrar todos los nodos del sistema.
- Comunicacion Estructurada: algoritmo de difusión en un sistema de redes de sensores.
- Comunicacion No Estructurada: redes P2P (peer-to-peer), donde los nodos pueden enviar mensajes directamente a cualquier otro nodo en la red sin que exista una estructura jerárquica predefinida, como en aplicaciones de intercambio de archivos como BitTorrent.
- Estatica: un sistema de colas de mensajes donde los productores siempre envían datos a las mismas colas y los consumidores siempre las leen de manera fija.
- Dinamica: un sistema de computación distribuida que utiliza un enfoque de "publicar-suscribir", donde los productores envían mensajes a diferentes suscriptores dependiendo de la carga de trabajo actual o el estado del sistema.
- Comunicacion Sincrona: protocolo de comunicación de un servidor y su cliente.
- Comunicacion Asincrona: uso de buffers circulares o colas de mensajes en sistemas distribuidos,

Descomposicion Recursiva - (Agrego Paso a Paso Quicksort)

Quicksort es un algoritmo de ordenamiento que sigue esta estrategia:

- Dividir: Elegir un elemento como pivote y dividir el arreglo en dos subarreglos:
    - Elementos mayores al pivote.
    - Elementos menores al pivote.
- Resolver recursivamente: Aplicar quicksort a cada subarreglo.
- Combinar: Unir los subarreglos ya ordenados junto con el pivote.

Ejercicio 2:

N°9 - ¿Qué es el balanceo de carga, y cuál es su importancia?

El **balanceo de carga** es una técnica utilizada para distribuir eficientemente el tráfico de red o la carga de procesamiento entre múltiples servidores o unidades de cómputo. Su objetivo es mejorar la disponibilidad, la escalabilidad y el rendimiento del sistema, evitando la sobrecarga de un solo recurso. Su importancia recae en;

- **Mayor disponibilidad:** Evita que un solo servidor se convierta en un punto único de falla.
- **Mejor rendimiento:** Distribuye las solicitudes de manera equitativa, reduciendo la latencia.
- **Escalabilidad:** Permite agregar o quitar recursos según la demanda sin interrumpir el servicio.
- **Optimización de recursos:** Asegura que ningún servidor esté subutilizado o sobrecargado.

N°14 - ¿Qué mecanismos pueden usarse para comunicar y sincronizar threads?

Para que los threads puedan coordinarse y compartir información de manera segura, se pueden utilizar los siguientes mecanismos:

1. **Memoria compartida:** Los threads pueden acceder a variables y estructuras de datos compartidas.
2. **Mensajes y colas:** Se pueden usar colas de mensajes para enviar datos entre threads.
3. **Pipes y sockets:** Utilizados cuando la comunicación debe realizarse entre procesos diferentes.

Además, a la hora de sincronizar los threads poseemos diferentes herramientas;

1. **Mutex (Mutual Exclusion):** Un mecanismo para garantizar que solo un thread acceda a un recurso compartido a la vez.
2. **Semáforos:** Controlan el acceso a recursos con múltiples permisos simultáneos.
3. **Monitores (Locks):** Encapsulan los mecanismos de sincronización dentro de estructuras de datos.
4. **Barrier (Barrera):** Hace que todos los threads esperen hasta que todos alcancen un punto de sincronización.
5. **Condiciones de espera (Condition Variables):** Permiten que un thread espere hasta que otro thread le notifique.

N°11 - ¿Qué mecanismo se puede utilizar para crear un proceso a partir de otro?

El mecanismo más común para crear un proceso a partir de otro es el **sistema de llamadas** `fork()` en sistemas tipo Unix/Linux.El cuak crea un nuevo proceso (hijo) que es una copia exacta del proceso padre. Aunque mbos procesos continúan ejecutándose a partir del punto donde se llamó `fork()`, los identificadores de proceso (PID) son diferentes entre sí.