Resumen Temas Parcial 2

Concurrencia e Hilos

La concurrencia es la capacidad de un sistema para ejecutar múltiples tareas de manera simultánea, proporcionando la ilusión de que se realizan al mismo tiempo. Esto se logra mediante el uso de **hilos**, que son puntos de ejecución independientes dentro de un programa. Los hilos comparten el mismo espacio de memoria, pero cada uno tiene su propio contador de programa, registros y pila.

Los hilos permiten implementar paralelismo y evitar bloqueos. Por ejemplo, en un servidor web, diferentes hilos pueden atender múltiples solicitudes de usuarios, permitiendo que unos esperen por operaciones de entrada/salida (I/O) mientras otros siguen procesando. Sin embargo, compartir datos entre hilos requiere mecanismos de sincronización para evitar problemas como condiciones de carrera.

Locks

Un **lock** es una herramienta de sincronización utilizada para garantizar **mutua exclusión** en regiones críticas, evitando que más de un hilo acceda a recursos compartidos al mismo tiempo. Un lock tiene dos operaciones básicas:

- **`lock()`**: Adquiere el control del lock si está disponible; si no, el hilo espera.
- **`unlock()`**: Libera el lock, permitiendo que otros hilos accedan a la región crítica.

Los locks son esenciales para proteger **secciones críticas**, como operaciones sobre variables compartidas. Por ejemplo, incrementar una variable global por varios hilos puede generar resultados inconsistentes si no se utiliza un lock. Aunque efectivos, los locks pueden generar problemas como **deadlocks**, donde dos o más hilos quedan esperando mutuamente.

Estructuras Concurrentes Basadas en Locks

Las estructuras concurrentes utilizan locks para garantizar consistencia mientras se permite el acceso de múltiples hilos. Algunos ejemplos son:

1. **Colas Concurrentes**: Usadas en sistemas de producción y consumo donde varios productores y consumidores interactúan. Los locks protegen las operaciones de encolar y desencolar.
2. **Tablas Hash Concurrentes**: Utilizan locks para sincronizar accesos a segmentos de la tabla, balanceando eficiencia y seguridad.
3. **Pilas Concurrentes**: Se asegura que las operaciones `push` y `pop` sean atómicas para evitar inconsistencias.

Estas estructuras son fundamentales en sistemas que requieren alta concurrencia, como servidores web y bases de datos.

Variables de Condición

Las **variables de condición** son primitivas de sincronización usadas junto con locks para que los hilos esperen ciertas condiciones antes de continuar. Funcionan mediante dos operaciones principales:

- **`wait()`**: El hilo se duerme hasta que otra parte del programa lo despierte.
- **`signal()`**: Notifica a un hilo en espera que la condición ha cambiado.

Por ejemplo, en un sistema de productor-consumidor, un productor puede llamar a `wait()` si el buffer está lleno, mientras que un consumidor puede usar `signal()` para despertar al productor cuando hay espacio disponible.

Semáforos

Un **semáforo** es una herramienta de sincronización más general que los locks. Puede ser usado tanto para exclusión mutua como para sincronización. Se caracteriza por mantener un contador interno que controla el acceso a recursos. Hay dos tipos principales:

1. **Semáforos binarios**: Funcionan como locks, permitiendo acceso exclusivo a un hilo.
2. **Semáforos contadores**: Permiten que múltiples hilos accedan a un recurso compartido, hasta un límite especificado.

Por ejemplo, en un sistema de impresión con múltiples impresoras, un semáforo contador puede permitir que hasta n hilos accedan simultáneamente a las impresoras.

Problemas de Concurrencia

Los sistemas concurrentes presentan desafíos únicos, como:

1. **Condiciones de Carrera**: Ocurren cuando el resultado del programa depende del orden en que los hilos acceden a recursos compartidos. Esto puede llevar a errores impredecibles.
    - **Ejemplo**: Dos hilos incrementando una variable sin protección.
2. **Deadlocks**: Situación donde dos o más hilos quedan bloqueados esperando recursos que otros hilos mantienen.
3. **Inanición**: Un hilo no recibe acceso a un recurso porque otros monopolizan el sistema.

4. **Bloqueo Activo**: Cuando un hilo consume recursos del sistema al esperar activamente que un recurso se libere, en lugar de dormir.