Resumen parcial 2

Tema	Lectura Libro Guía	
Concurrencia e Hilos	25	26
Locks	27	28
Estructuras concurrentes basadas en Lock	29	
Variables de condición	30	
Semáforos	31	
Problemas de Concurrencia	32	34

Resumen: Introducción a la Concurrencia y Hilos

1. Conceptos Básicos de Concurrencia y Hilos

- **Concurrencia**: Permite que múltiples tareas se ejecuten de manera "simultánea". En sistemas con múltiples CPUs, esto puede mejorar significativamente el rendimiento.
- **Hilos (Threads)**:
    - Un **hilo** es una unidad básica de ejecución dentro de un proceso.
    - Los hilos dentro de un proceso comparten el mismo espacio de direcciones, lo que facilita el acceso a datos compartidos pero también puede causar problemas como **condiciones de carrera**.

2. Diferencias entre Procesos e Hilos

- **Procesos**: Son independientes, con su propio espacio de memoria y recursos.
- **Hilos**: Comparten el espacio de memoria del proceso principal, lo que permite la comunicación más eficiente pero introduce riesgos de sincronización.

3. Estructura de un Hilo

- Cada hilo tiene:
    - Su propio **contador de programa** (PC).
    - Un conjunto de **registros**.
    - Una pila (stack) separada, mientras comparten el heap y el código del programa.

4. Ventajas del Uso de Hilos

1. **Paralelismo**: Los hilos pueden dividir tareas grandes entre múltiples CPUs para acelerar los cálculos.
2. **Evitación de Bloqueos**: Mientras un hilo está bloqueado (por ejemplo, esperando una operación de E/S), otros hilos pueden continuar ejecutándose, maximizando el uso de la CPU.

5. Creación de Hilos: Ejemplo en C

```
#include <pthread.h>
void *mythread(void *arg) {
   printf("%s\n", (char *) arg);
   return NULL;
int main() {
   pthread_t p1, p2;
   printf("main: begin\n");
   pthread_create(&p1, NULL, mythread, "A");
   pthread_create(&p2, NULL, mythread, "B");
   pthread_join(p1, NULL);
   pthread_join(p2, NULL);
   printf("main: end\n");
   return 0;
}
```

- Este programa crea dos hilos, cada uno imprime un mensaje y luego finaliza.

6. Problemas Comunes con Hilos

- **Condiciones de Carrera**: Ocurren cuando múltiples hilos acceden o modifican datos compartidos simultáneamente, generando resultados no determinísticos.
- **Sección Crítica:** Parte del código que accede a recursos compartidos y debe ejecutarse de manera exclusiva.
- **Exclusión Mutua:** Uso de mecanismos (como cerrojos) para garantizar que solo un hilo acceda a la sección crítica a la vez.

7. **Ejemplo de Condición de Carrera**

- **Código para incrementar una variable compartida:**

```
static volatile int counter = 0;

void *mythread(void *arg) {
   for (int i = 0; i < 1e7; i++) {
      counter++;
   }
   return NULL;
}
```

Resultado esperado: `counter = 20000000`. Sin embargo, debido a las interrupciones y la falta de sincronización, los resultados pueden variar entre ejecuciones

Soluciones para Sincronización

- **Operaciones Atómicas**: Garantizan que las operaciones críticas se ejecuten completamente sin interrupciones.
- **Primitivas de Sincronización**:
    - **Mutexes**: Bloqueos para garantizar que solo un hilo entre a la sección crítica.
    - **Semáforos**: Permiten gestionar acceso a recursos limitados.
    - **Variables de Condición**: Usadas para coordinar hilos.

9. Importancia en los Sistemas Operativos

- La concurrencia es esencial para manejar múltiples procesos y tareas en un sistema operativo.
- Ejemplo: Actualización de estructuras como tablas de páginas o listas de procesos en un sistema multitarea.

10. Conceptos Clave

- **Condición de Carrera**: Resultado impredecible debido a hilos concurrentes accediendo a datos compartidos.
- **Sección Crítica**: Código que requiere exclusión mutua para evitar conflictos.
- **Atomicidad**: Garantía de que una operación ocurra completa o no ocurra en absoluto.

Introducción al API de Hilos (Thread API)

1. Introducción al API de Hilos

- Un **API de hilos** es un conjunto de herramientas y funciones que permite crear y gestionar hilos en un programa.
- Se cubren tres áreas principales:
    - **Creación de hilos**.
    - **Sincronización y exclusión mutua**.
    - **Variables de condición** para coordinar acciones entre hilos.

- El **API de hilos** proporciona las herramientas necesarias para manejar concurrencia.
- Claves para usarlo eficientemente:
    - Crear y gestionar hilos con cuidado.
    - Sincronizar acceso a recursos compartidos con **mutexes**.
    - Coordinar acciones entre hilos con **variables de condición**.

Resuemn Locks

Bloqueos (Locks)

1**. Introducción a los Bloqueos**

Los bloqueos son mecanismos para garantizar la exclusión mutua en secciones críticas, asegurando que solo un hilo pueda ejecutarlas a la vez.

Se utilizan para evitar condiciones de carrera cuando varios hilos acceden a recursos compartidos.

2**. Concepto Básico de un Bloqueo**

Un bloqueo tiene dos estados: disponible o adquirido.

La función lock() asegura la exclusión mutua al bloquear otros hilos hasta que se complete la sección crítica.

La función unlock() libera el bloqueo para que otros hilos puedan acceder.

Implementación con POSIX

- En POSIX, los bloqueos se implementan como **mutexes**:

Evaluación de los Bloqueos

Correctitud: Deben garantizar exclusión mutua.

Equidad: Todos los hilos deben tener la misma oportunidad de acceder al recurso.

Desempeño: Evaluar el impacto en diferentes casos (sin contención, con contención en un solo CPU y en múltiples CPUs).

5**. Métodos de Construcción de Bloqueos**

Deshabilitar Interrupciones:

Usado en sistemas con un solo procesador.

Ineficiente en multiprocesadores y arriesgado (puede causar monopolización del procesador).

Spin Locks:

Los hilos esperan ocupando ciclos de CPU.

Bloqueos con Yield:

Los hilos ceden el CPU si no pueden adquirir el bloqueo, reduciendo el desperdicio de ciclos.

Ticket Locks:

Garantizan equidad asignando un "turno" a cada hilo.

Problemas Clásicos en Bloqueos

1. **Spin-Waiting**: Consume CPU innecesariamente.
2. **Starvation**: Algunos hilos podrían no adquirir el bloqueo.
3. **Inversión de Prioridades**: Un hilo de alta prioridad queda esperando mientras uno de baja prioridad mantiene el bloqueo.

En resumen

- Los bloqueos son esenciales para manejar concurrencia, pero deben implementarse cuidadosamente para equilibrar correctitud, equidad y desempeño.
- Métodos modernos combinan soporte de hardware y del sistema operativo para optimizar el uso de bloqueos.

Variables de Condición

- Herramienta de sincronización que permite a los hilos esperar hasta que una condición específica sea verdadera.
- Operaciones principales:
    - **wait()**: Pone un hilo en espera hasta que se cumpla la condición.
    - **signal()**: Despierta un hilo en espera cuando la condición cambia.

Problema Productor/Consumidor

- Clásico problema de sincronización:
    - **Productores**: Generan datos y los colocan en un buffer.
    - **Consumidores**: Consumen los datos del buffer.
- Requiere sincronización para evitar que:
    - Un productor intente agregar datos en un buffer lleno.
    - Un consumidor intente sacar datos de un buffer vacío.

Solución Correcta

- **Uso de mutexes** y **dos variables de condición**:
    - Una para controlar cuándo el buffer está lleno.
    - Otra para controlar cuándo el buffer está vacío.
- **Implementación clave:**
    - Productores esperan cuando el buffer está lleno.
    - Consumidores esperan cuando el buffer está vacío.
    - Siempre se utiliza while para reevaluar las condiciones antes de actuar.

Lecciones Clave

Siempre usar while en lugar de if para verificar condiciones, evitando problemas.

Usar dos variables de condición mejora la eficiencia al despertar solo a los hilos relevantes (productores o consumidores).

1. ¿Qué son los Semáforos?

- Los semáforos son herramientas de sincronización utilizadas en programación concurrente para controlar el acceso a recursos compartidos.
- Son contadores que pueden ser incrementados (señal) o decrementados (espera), dependiendo de las operaciones de los hilos.

Tipos de Semáforos

1. **Semáforos Binarios**:
    - Actúan como un interruptor (0 o 1).
    - Similares a los mutexes, se utilizan para exclusión mutua.
2. **Semáforos de Conteo**:
    - Permiten múltiples accesos simultáneos a un recurso, dependiendo del valor del contador.

Operaciones Principales

1. **wait()**:
    - Decrementa el contador.
    - Si el valor del contador es menor a cero, el hilo se bloquea.
2. **signal()**:
    - Incrementa el contador.
    - Si hay hilos en espera, uno de ellos es despertado.

Problemas que Resuelven los Semáforos

- **Control de acceso**: Limita cuántos hilos pueden acceder a un recurso compartido.
- **Sincronización**: Garantiza que ciertas acciones ocurran en orden, evitando condiciones de carrera.

Ejemplos:

Los semáforos se utilizan para sincronizar hilos productores y consumidores:

- **Semáforo "vacío"**: Indica cuántos espacios libres hay en el buffer.
- **Semáforo "lleno"**: Indica cuántos elementos hay disponibles en el buffer.
- **Mutex**: Protege el acceso al buffer para evitar condiciones de carrera.

Concurrencia

1. Tipos de Problemas Comunes

- **Violaciones de Atomicidad**: Ocurren cuando una secuencia de instrucciones que debe ejecutarse de forma atómica no lo hace, lo que causa errores impredecibles.
- **Violaciones de Orden**: Suceden cuando el orden esperado entre operaciones de distintos hilos no se cumple.

Deadlocks (Interbloqueos)

- Se producen cuando dos o más hilos esperan indefinidamente por recursos que están bloqueados por otros hilos.
- **Condiciones para un Deadlock**:
    1. Exclusión mutua.
    2. Retención y espera.
    3. No expropiación.
    4. Espera circular.

Soluciones y Prevención

- **Evitar Deadlocks**:
    - Diseñar un orden estricto en la adquisición de locks.
    - Utilizar técnicas como trylock para evitar bloqueos.
- **Sincronización**:
    - Usar variables de condición para asegurar un orden correcto entre hilos.
    - Emplear estructuras sin bloqueo (lock-free) para evitar problemas de concurrencia.

Buenas Prácticas

- Analizar cuidadosamente el diseño de los hilos y su interacción.
- Priorizar soluciones robustas como el uso de locks bien estructurados o herramientas modernas para evitar errores.