ITESO

Universidad Jesuita
de Guadalajara

Axel Roberto Orozco Hernández
NAVARRO QUINN, LUIS ROBERTO

Fundamentos de Sistemas Operativos
Profesor: Leonardo Sandoval. Gonzalez

Actividad 15

1.- Del problema de los 4 jugadores de dominó sentados en una mesa jugando dominó, se van turnando para tirar una ficha o pasar si no pueden jugar en el sentido contrario a las manecillas del reloj. Revisa la solución en Java que está en la carpeta del curso.

- ¿Cuál es el monitor?

En el problema de los cuatro jugadores de domino. El monitor es el mecanismo de sincronización que controla el acceso de los recursos compartidos, en este caso, las reglas del turno para jugar. El Monitor asegura que sólo un jugador pueda hacer su jugada a la vez y coordina el paso del turno al siguiente jugador.

- ¿Cuantas colas de condición hay en un monitor en Java?

En Java, un monitor puede tener múltiples colas de condición. Cada cola de condición está asociada con un objeto de bloqueo y puede tener múltiples y los esperando en ella, bloqueados en una llamada a 'Wait()'

- Si en Java ejecutas la sentencia notifyAll() se desbloquean todos los hilos que están bloqueados en la sentencia wait(), explica por qué solamente uno que es el siguiente jugador es el que continua.

Cuándo se ejecuta notifyAll(), se despiertan todos los hilos que están esperando en la sentencia Wait(). Sin embargo, sólo un hilo puede reanudar la ejecución en el monitor, ya que el monitor permite la entrada de un hilo a la vez. El siguiente hilo en el turno continúa, mientras que los demás permanecen bloqueados hasta que pueda entrar el monitor.

2.- En el caso del problema del productor consumidor que se muestra a continuación, la sincronización se hace con un monitor con notificación definido a partir de semáforos.

Definición de funciones usando semáforos:

1. Declarar los semáforos globalmente para gestionar la entrada y la sincronización de hilos.

```
sem_t sem_monitor;
sem_t sem_condition;
int waiting = 0;
```

2. implementacion de enter_monitor() para que solo un hilo entre al monitor.

```
Void enter_monitor(){
    Sem_wait(&sem_monitor);
```

```
}
```

3. Implementacion de leave_monitor() para liberar el monitor para otro hilo.

```
Void leave_monito() {
    Sem_post(&sem_monitor);
}
```

4. Implementacion de cwait() Bloque el hilo en una cola de condicion.

```
Void cwait() {
    Waiting ++;
    Sem_post(&sem_monitor);
    Sem_post(&sem_conidition);
    Sem_post(&sem_monitor);
    Waiting--;
}
```

5. Implementacion de cnotify() que desbloquea un hilo que esta en la cola de condicion.

```
Void cnotify(){
    If( waiting > 0){
            Sem_post(&sem_condition);
    }
}
```

Se tiene que modificar el código para usar las funciones de monitor
El código proporcionado para buffer_put() y buffer_get() debe usar las funciones de monitor para manejar la exclusión mutua y la sincronización.

¿Qué aprendieron?

Monitores y sincronización: aprendimos cómo implementar un monitor usando semáforos en C, entendiendo las funciones básicas que permite encontrar el acceso concurrente a recursos compartidos.

Semáforos para la exclusión mutua: usar semáforos para controlar la entrada a una sección crítica es crucial para evitar condiciones de carrera y asegurar que ser un proceso acceda al recurso convertido a la vez.

Colas de condición: las colas de condición permiten bloquear procesos de manera ordenada y con cnotify(), es posible despertar un proceso de la cola, asegurando que los recursos se utilizan de manera eficiente.