Cuestionario teoría 3 a 4

sábado, 10 de octubre de 2020 18:30

1- ¿Por qué las propiedades de vida dependen de la política de scheduling? ¿Cómo aplicaría el concepto de fairness al acceso a una base de datos compartida por n procesos concurrentes?

Las propiedades de vida dependen de la política de scheduling porque si se aplica una mala política de scheduling nos puede llevar a un deadlock, que es lo que la propiedad de vida intenta evitar.
El concepto de fairness se puede aplicar al acceso a una base de datos proponiendo que se debe garantizar la propiedad de fairness para que todos los procesos puedan acceder.

2- Dado el siguiente programa concurrente, indique cuál es la respuesta correcta (justifique claramente)

```
int a = 1, b = 0;
co ⟨await (b = 1) a = 0 ⟩ // while (a = 1) { b = 1; b = 0; } oc
```

a) Siempre termina
b) Nunca termina
c) Puede terminar o no

c) Puede pasar que cuando el segundo proceso haga b=1 el primer proceso lea el await y haga sus tareas, lo que lleva a que el programa termine. Pero también puede pasar que cada vez que el await chequee la condición b está en 0, porque coincide con la segunda operación del otro proceso.

3- ¿Qué propiedades que deben garantizarse en la administración de una sección crítica en procesos concurrentes? ¿Cuáles de ellas son propiedades de seguridad y cuáles de vida?
En el caso de las propiedades de seguridad, ¿cuál es en cada caso el estado "malo" que se debe evitar?

Propiedades a cumplir		
Exclusión mutua: A lo sumo un proceso está en su SC	Seguridad	Se debe evitar un estado inconsistente que se puede dar si dos procesos entran en la SC
Ausencia de Deadlock (Livelock): si 2 o más procesos tratan de entrar a sus SC, al menos uno tendrá éxito.	Vida	
Ausencia de Demora Innecesaria: si un proceso trata de entrar a su SC y los otros están en sus SNC o terminaron, el primero no está impedido de entrar a su SC.	Vida	
Eventual Entrada: un proceso que intenta entrar a su SC tiene posibilidades de hacerlo (eventualmente lo hará).	Vida	

4- Resuelva el problema de acceso a sección crítica para N procesos usando un proceso coordinador. En este caso, cuando un proceso SC[i] quiere entrar a su sección crítica le avisa al coordinador, y espera a que éste le otorgue permiso. Al terminar de ejecutar su sección crítica, el proceso SC[i] le avisa al coordinador. Desarrolle una solución **de grano fino** usando únicamente variables compartidas (ni semáforos ni monitores).

```
int arribo[n] = 0;
int continuar[n] = 0;

process Worker[id=1 to n]{
    arribo[i] = 1;
    while(continuar[i] == 0){}
    //hace la sección crítica
    continuar[i] = 0;
}

process Coordinador{
    actual = 0;
    while(True){
            while(arribo[actual] = 0){
                actual = (actual + 1) mod n
            }
            continuar[actual] = 1;
            while(continuar[actual] = 1){}
        }
    }
}
```

5- ¿Qué mejoras introducen los algoritmos Tie-breaker, Ticket o Bakery en relación a las soluciones de tipo spin-locks?

La ausencia de la necesidad de instrucciones especiales

6- Modifique el algoritmo Ticket para el caso en que no se dispone de una instrucción Fetch and Add

```
int numero = 1, proximo = 1, turno[1:n] = ( [n] 0 );

process SC [i: 1..n]{
    while (true){
            while (TS(lock)) skip ;
            turno[i] = numero;
            numero = numero + 1;
            lock = false;
            //turno[i] = FA (numero, 1);
            while (turno[i] <> proximo)skip;
            sección crítica;
            proximo = proximo + 1;
            sección no crítica;
        }
}
```

7- Analice las soluciones para las barreras de sincronización desde el punto de vista de la complejidad de la programación y de la performance.

Contador Compartido: es muy simple en cuanto a complejidad, pero genera un problema al tener que reiniciar el contador 0 para la siguiente iteración.
Flags y Coordinadores: sigue la idea del contador compartido, es más complejo porque requiere la interacción entre dos procesos pero no tenemos el problema de reiniciar el contador a 0.
Árboles: es complejo porque requiere un proceso extra, un procesador extra y una complejidad agregada para armar el arbol, pero la performance se incrementa considerablemente. El tiempo de ejecución del coordinador es proporcional a n.
Barreras Simétrica(Butterfly barrier): lleva $\log_2 n$ etapas: cada worker sincroniza con uno distinto en cada etapa. Es el de mejor performance pero el más complejo al mismo tiempo.

9- a) Explique la semántica de un semáforo.
b) Indique los posibles valores finales de *x* en el siguiente programa (**justifique claramente su respuesta**):

```
int x = 4; sem s1 = 1, s2 = 0;
co P(s1); x = x * x ; V(s1);
    // P(s2); P(s1); x = x * 3; V(s1):
    // P(s1); x = x - 2; V(s2); V(s1);
oc
```

a)

Semáforo ⇒ instancia de un tipo de datos abstracto (o un objeto) con sólo 2 operaciones (métodos) *atómicas*: ***P*** y ***V***.

Internamente el valor de un semáforo es un entero *no negativo:*

- ***V*** → Señala la ocurrencia de un evento (incrementa).
- ***P*** → Se usa para demorar un proceso hasta que ocurra un evento (decrementa).

b)
X = 42
X = 12
X = 36

10- Desarrolle utilizando semáforos una solución centralizada al problema de los filósofos, con un administrador único de los tenedores, y posiciones libres para los filósofos (es decir, cada filósofo puede comer en cualquier posición siempre que tenga los dos tenedores correspondientes).

```
int tenedores = 5;
sem usarTenedores = 1;
sem comer = 0;

process Filosofos[i = 0..3]{
    while(true){
        V(hayGente);
        P(comer);
        comer;
        P(usarTenedores);
        tenedores = tenerdores + 2;
        V(usarTenedores);
        pensar;
    }
}

process coordinador{
    while(True){
        P(hayGente);
        P(usarTenedores);
        if(tenedores - 2 >= 0){
            tenedores = tenedores-2;
            V(usarTenedores);
            V(comer);
        } else {
            V(usarTenedores);
        }

    }
}
```

11- Describa la técnica de *Passing the Baton*. ¿Cuál es su utilidad en la resolución de problemas mediante semáforos?

La técnica passing the baton utilizan los semáforos binarios divididos para brindar exclusión y despertar los procesos demorados

Passing the baton: técnica general para implementar sentencias ***await.***

Cuando un proceso está dentro de una SC mantiene el *baton* (*testimonio*, *token*) que significa permiso para ejecutar.

Cuando el proceso llega a un ***SIGNAL*** (sale de la SC), pasa el *baton* (control) a otro proceso. Si ningún proceso está esperando por el *baton* (es decir esperando entrar a la SC) el *baton* se libera para que lo tome el próximo proceso que trata de entrar.