Programación Concurrente 2024

Cuestionario guía - Clases Teóricas 3, 4 y 5

1- ¿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 de Seguridad (Safety):

Exclusión Mutua: Garantiza que, como máximo, un proceso está en su sección crítica en un momento dado. Se debe evitar que dos o más procesos estén simultáneamente en la misma sección crítica. El estado "malo" a evitar es cuando múltiples procesos intentan ejecutar la sección crítica al mismo tiempo, lo que podría causar conflictos y resultados incorrectos.

Propiedades de Vida (Liveness):

Ausencia de Deadlock (Livelock): Garantiza que, si dos o más procesos intentan entrar en sus secciones críticas, al menos uno de ellos tendrá éxito. Se debe evitar que un proceso espere indefinidamente a que se libere un recurso que nunca se libera porque otros procesos también lo están esperando. El "estado malo" es cuando los procesos quedan bloqueados en un punto muerto y no pueden avanzar.

Ausencia de Demora Innecesaria: Asegura que si un proceso intenta entrar en su sección crítica y los otros procesos están en secciones no críticas o ya han terminado, el primer proceso no debe ser impedido de entrar en su sección crítica. Se debe evitar que un proceso quede esperando innecesariamente por un recurso que ya está disponible. El "estado malo" es cuando un proceso está bloqueado a pesar de que los recursos están disponibles.

Eventual Entrada: Garantiza que un proceso que intenta entrar en su sección crítica tiene la posibilidad de hacerlo en algún momento (eventualmente lo hará). Se debe evitar la inanición, donde un proceso nunca obtiene acceso a su sección crítica debido a la priorización constante de otros procesos. El "estado malo" es cuando un proceso queda excluido repetidamente de su sección crítica.

2- 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 actual = -1;
process sc[i: 0..n-1]{
  while (true){
    while(actual <> i) skip;
       //SC
       listo = true;
    while(listo) skip;
    }
  }
process coordinador{
while(true){
     for j = 0..n-1 {
        actual = j;
        while(!listo) skip;
           listo = false;
        }
     }
}
```

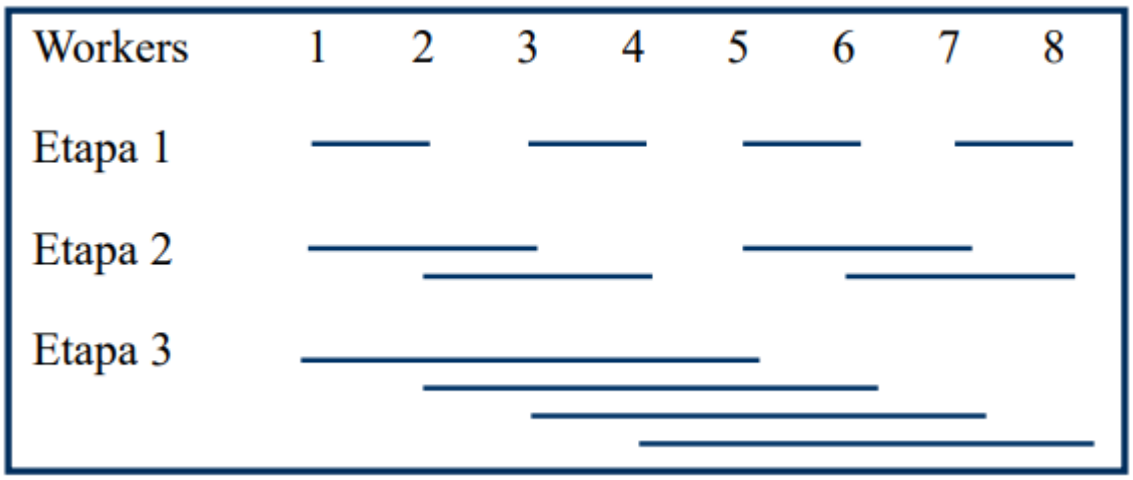
3- ¿Qué mejoras introducen los algoritmos Tie-breaker, Ticket o Bakery en relación a las soluciones de tipo spin-locks?

Los spin-locks no controlan el orden de los procesos demorados es posible que alguno no entre nunca si el scheduling no es fuertemente fair (race conditions).
Los algoritmos Tie-breaker, Ticket y Bakery introducen mejoras al proporcionar equidad en la asignación de recursos y prevenir la inanición de procesos.

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

En las barreras que involucran un coordinador, se añade la necesidad de un proceso adicional, lo que implica un costo adicional tanto en términos de recursos de procesamiento como de tiempo de ejecución el cual es proporcional a n. Estas barreras son relativamente simples de implementar y pueden funcionar bien cuando se trabaja con un número reducido de procesos, donde la sobrecarga de comunicación entre el coordinador y los trabajadores no tiene un impacto significativo en comparación con el tiempo de espera. Sin embargo, a medida que el número de procesos aumenta, el coordinador puede convertirse en un cuello de botella, lo que podría afectar de manera negativa el rendimiento global del sistema.
Una posible solución al problema de rendimiento de la barrera en donde se tiene un coordinador es combinar las acciones de Workers y Coordinador, haciendo que cada Worker sea también Coordinador. En estos casos se tienen Workers en forma de árbol en donde las señales de arribo van hacia arriba en el árbol, y las de continuar hacia abajo. Este tipo de barreras pueden tener un rendimiento mejorado en sistemas con un gran número de procesos. Sin embargo, puede requerir un diseño más elaborado para garantizar que la sincronización se realice correctamente.

5- Explique gráficamente cómo funciona una butterfly barrier para 8 procesos usando variables compartidas.



6- a) Explique la semántica de un semáforo.

Es una 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).

- Semáforo general (o counting semaphore)

  P(s): < await (s > 0) s = s-1; >
  V(s): < s = s+1; >

- Semáforo binario
  P(b): < await (b > 0) b = b-1; >
  V(b): < await (b < 1) b = b+1; >

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
```

Los valores finales posibles de x son 36 o 42
Si se hace en el siguiente orden:

1 - P(s1);
2 - x = x * x ;
3 - V(s1);

1 - P(s1);
1 - x = x - 2;
2 - V(s2);
3 - V(s1);

8- P(s2);
9- P(s1);
10- x = x * 3;
11- V(s1):

El valor será 42 (podría incluso hacerse 1, 2, 3, 4, 5, 6, 8, 9, 7, 9, 10, 11 y el valor seria el mismo, ya que el proceso se va a quedar bloqueado en 4 esperando a que s1 sea > 0).
Si se hace

1 - P(s1);
2 - x = x - 2;
3 - V(s2);
4 - V(s1);

1 - P(s2);
2 - P(s1);
3 - x = x * 3;
4 - V(s1);

9- P(s1);
10- x = x * x ;
11- V(s1);

El valor será 36 (podría incluso hacerse 1, 2, 3, 5, 6, 4, 6, 7, 8, 9, 10, 11 y el valor seria el mismo, ya que el proceso se va a quedar bloqueado en 6 esperando a que s1 sea > 0).

Además de estos dos valores finales, no hay ninguno mas ya que no son posibles de obtener puesto que las

operaciones que se están haciendo sobre x logran ser atómicas gracias al uso de semáforos. Si los procesos se intercalaran de otra manera, se quedarían esperando en los semáforos hasta poder avanzar y el resultado de X seria uno de los dos valores ya mencionados.

7- 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). EXCLUSION MUTUA SELECTIVA.

```
sem  tenedores [5] = {1,1,1,1,1};

process Filososfos[i = 0..3]
{ while(true)
    { P(tenedor[i]);  P(tenedor[i+1]);
      comer;
      V(tenedor[i]);  V(tenedor[i+1]);
    }
}

process Filososfos[4]
{ while(true)
    { P(tenedor[0]);  P(tenedor[4]);
      comer;
      V(tenedor[0]);  V(tenedor[4]);
    }
}
```

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

Passing the Baton es una técnica general utilizada para implementar sentencias await. En esta técnica, un proceso que necesita acceso a una sección crítica mantiene el "bastón" o "testimonio" que le otorga el permiso para ejecutar. Una vez que un proceso ha finalizado su tarea o ha salido de su sección crítica, pasa el bastón al siguiente proceso en la secuencia. Si no hay procesos esperando el bastón (es decir esperando entrar a la SC), este se libera para que lo tome el próximo proceso que trata de entrar. La utilidad principal de esta técnica radica en garantizar que los procesos se ejecuten de manera ordenada y sin interferencias entre ellos.

9- Modifique las soluciones de Lectores-Escritores con semáforos de modo de no permitir más de 10 lectores simultáneos en la BD y además que no se admita el ingreso a más lectores cuando hay escritores esperando.

Se debe reescribir en la primera condición de los lectores así
if (nw > 0 || dw > 0) para que no se admita el ingreso de lectores si hay escritores esperando.
Para no permitir más de 10 lectores simultáneos, se puede usar un semáforo contador de recursos. El código de los lectores quedaría

```
sem mutexR = 10; // Semaforo contador de lectores en la BD.
process Lector [i = 1 to M] {
while(true) {
P(mutexR)
P(e);
if (nw > 0 || dw > 0) { // Si hay escritores esperando o en BD se duerme
```

```
dr = dr+1;
V(e);
P(r);
}
nr = nr + 1;
if (dr > 0) {
dr = dr – 1;
V(r);
}
else V(e);
lee la BD;
P(e);
nr = nr – 1;
if (nr == 0 and dw > 0) {
dw = dw – 1;
V(w);
}
else V(e);
V(mutexR)
}
}
```

10- Describa el funcionamiento de los monitores como herramienta de sincronización.

Los monitores son módulos de programa con más estructura, y que pueden serimplementados tan eficientemente como los semáforos. Son un mecanismo deabstracción de datos:

Encapsulan las representaciones de recursos.
Brindan un conjunto de operaciones que son los únicos medios para manipula resos recursos.

Contiene variables que almacenan el estado del recurso y procedimientos que implementan las operaciones sobre él.

La exclusión mutua está dada por los monitores. Es implícita. Un monitor va a estar siendo utilizado en un momento solo por UN proceso. Los demás se encolan y cuandoel proceso termine de ser usado, se va a elegir uno encolado de forma no determinística.

En los programas concurrentes con monitores hay procesos activos y monitorespasivos. Dos procesos interactúan invocando procedures de un monitor.

Ventajas:

Un proceso que invoca un procedure puede ignorar cómo está implementado.
El programador del monitor puede ignorar cómo o dónde se usan los procedures.

Los monitores tienen interfaz y cuerpo. Sólo los nombres de los procedures (la interfaz)son visibles desde afuera. Los procedures pueden acceder sólo a variables permanentes, sus variables locales, y parámetros que le sean pasados en la invocación. La comunicación entre procesos será haciendo uso del monitor que va a contener los datos compartidos, que son variables o estructuras de datos accesibles por varios procesos. Dentro del monitor van a haber variables condicionales que permitirán a los procesos esperar o notificar eventos específicos.

La sincronización por condición es con variables condición: cond cv;
cv es una cola de procesos demorados, no visible directamente al programador.

wait(cv): el proceso se demora al final de la cola de cv y deja el acceso exclusivo almonitor.
signal(cv): despierta al proceso que está al frente de la cola (si hay alguno) y lo sacade ella. El proceso despertado recién podrá ejecutar cuando readquiera el acceso exclusivo al monitor.
signal_all(cv): despierta todos los procesos demorados en cv, quedando vacía lacola asociada a cv.

Operaciones adicionales que NO SON USADAS EN LA PRÁCTICA sobre las variablescondición:

empty(cv): retorna true si la cola controlada por cv está vacía.

Solución: usar una variable contadora de la cantidad de procesos dormido.Cuando un proceso se va a dormir incrementa una variable: ej: espera ++. El proceso que se encarga de despertarlo la decrementa.

wait(cv, rank): el proceso se demora en la cola de cv en orden ascendente deacuerdo al parámetro rank y deja el acceso exclusivo al monitor.

Solución: utilizo una cola.

minrank(cv): función que retorna el mínimo ranking de demora.

Solución: utilizo una cola ordenada de min a max y siempre que hagosignal(cv) despierto al mínimo.

11- ¿Qué diferencias existen entre las disciplinas de señalización “Signal and wait” y “Signal and continue”?

La diferencia principal entre ambas disciplinas recae en quien es el que va a serencolado nuevamente para esperar el uso del monitor. En el caso de Signal and continued va a ser encolado (en la cola no determinística) el proceso despertado y enSignal and wait va a ser encola el proceso que despierta.

Signal and continued

El proceso que hace el signal continúa usando el monitor, y el proceso despertado pasaa competir por acceder nuevamente al monitor para continuar con su ejecución (en la instrucción que lógicamente le sigue al wait).

Signal and wait.

El proceso que hace el signal pasa a competir por acceder nuevamente al monitor, mientras que el proceso despertado pasa a ejecutar dentro del monitor a partir deinstrucción que lógicamente le sigue al wait.

12- ¿En qué consiste la técnica de Passing the Condition y cuál es su utilidad en la resolución de problemas con monitores? ¿Qué relación encuentra entre passing the condition y passing the baton?

Passing The Condition se refiere a la transferencia de la condición que se está esperando de un proceso a otro. La idea central es permitir que un proceso notifique aotro sobre un cambio en una condición específica, lo que generalmente se realiza mediante el uso de variables condicionales.

Si hay un proceso dormido esperando por determinada condición, el que lo estababloqueando lo despierta, básicamente le pasa la condición para que pueda seguir ejecutándose. Si no hay ningún proceso bloqueado, se hará verdadera la condiciónpara que el próximo proceso que venga pueda usarla. La idea central es que un proceso notifique a otro sobre un cambio en una condición específica
La utilidad en la resolución de problemas con monitores es que se respete el orden deespera de los procesos.

La relación entre passing the condition y passing the baton es que ambas técnicas se basan en el concepto de transferir el uso o acceso de recursos críticos entre procesos,
aunque passing the condition se centra más en la sincronización haciendo uso de condiciones y passing the baton en la transferencia de control de un proceso a otro.

13- Desarrolle utilizando monitores 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).

14- Sea la siguiente solución propuesta al problema de alocación SJN:

```
monitor SJN {
        bool libre = true;
        cond turno;

        procedure request(int tiempo) {
                if (not libre) wait(turno, tiempo);
                libre = false;
         }

        procedure release() {
                libre = true
                signal(turno);
         }
}
```

a) Funciona correctamente con disciplina de señalización Signal and Continue?
b) Funciona correctamente con disciplina de señalización Signal and Wait?
EXPLIQUE CLARAMENTE SUS RESPUESTAS

15- Modifique la solución anterior para el caso de no contar con una instrucción wait con prioridad.

16- Modifique utilizando monitores las soluciones de Lectores-Escritores de modo de no permitir más de 10 lectores simultáneos en la BD, y además que no se admita el ingreso a más lectores cuando hay escritores esperando.

17- Resuelva con monitores el siguiente problema: Tres clases de procesos comparten el acceso a una lista enlazada: searchers, inserters y deleters. Los searchers sólo examinan la lista, y por lo tanto pueden ejecutar concurrentemente unos con otros. Los inserters agregan nuevos ítems al final de la lista; las inserciones deben ser mutuamente exclusivas para evitar insertar dos ítems casi al mismo tiempo. Sin embargo, un insert puede hacerse en paralelo con uno o más searches. Por último, los deleters remueven ítems de cualquier lugar de la lista. A lo sumo un deleter puede acceder la lista a la vez, y el borrado también debe ser mutuamente exclusivo con searches e inserciones.

18- El problema del "Puente de una sola vía" (One-Lane Bridge): autos que provienen del Norte y del Sur llegan a un puente con una sola vía. Los autos en la misma dirección pueden atravesar el puente al mismo tiempo, pero no puede haber autos en distintas direcciones sobre el puente.
   a) Desarrolle una solución al problema, modelizando los autos como procesos y sincronizando con un monitor (no es necesario que la solución sea fair ni dar preferencia a ningún tipo de auto).
   b) Modifique la solución para asegurar fairness (Pista: los autos podrían obtener turnos).