Cuestionario guía - Clases Teóricas 3 y 4

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 se centran en garantizar que los procesos puedan avanzar y completar sus tareas, evitando situaciones de bloqueo y asegurando que las acciones críticas se ejecuten eventualmente. Dependen de la política de scheduling porque esta última determina el comportamiento de la ejecución de procesos concurrentes, y diferentes políticas de scheduling pueden afectar la equidad y la capacidad de los procesos para avanzar y completar sus tareas.

El concepto de fairness se puede aplicar al acceso a una base de datos compartida por n procesos concurrentes de la siguiente manera:

Fairness Incondicional: En este contexto, significa garantizar que cada proceso tenga la oportunidad de acceder a la base de datos compartida en algún momento. Ningún proceso debería ser excluido indefinidamente del acceso a la base de datos.

Fairness Débil: Además de garantizar el acceso a la base de datos, también implica asegurarse de que todos los procesos tengan la oportunidad de realizar operaciones en la base de datos, independientemente de sus condiciones o restricciones. Por ejemplo, si un proceso espera una condición específica para acceder a la base de datos, la política de scheduling debe garantizar que, si la condición se cumple, el proceso pueda acceder a la base de datos.

Fairness Fuerte: Va un paso más allá y asegura que, si un proceso tiene una condición para acceder a la base de datos y esa condición se cumple en algún momento, el proceso obtendrá acceso a la base de datos con suficiente frecuencia, incluso si la condición es rara.

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 → Esta es la respuesta correcta. En el caso que se ejecutara siempre while (a = 1) { b = 1; b = 0 } y luego <await (b = 1) a = 0> repetidas veces, el programa nunca terminara porque el primer proceso se va a quedar bloqueado hasta que b sea 1 y esto no sucedera ya que tendrá el valor de 0. A su vez puede sucedes que se ejecute while (a = 1) b = 1, luego ⟨await (b = 1) a = 0 ⟩ y luego b = 0, en ese caso el programa terminara.

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 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.

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 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;
    }
  }
}
```

5- ¿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 $\Rightarrow$ es posible que alguno no entre nunca si el scheduling no es fuertemente fair (race conditions).

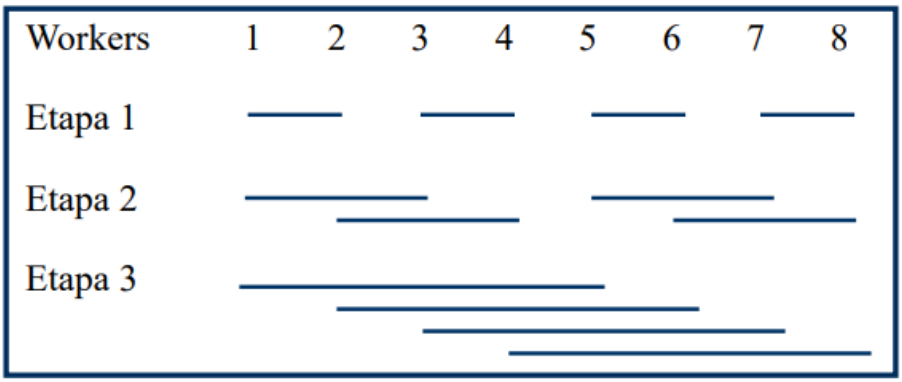
Los algoritmos Tie-breaker, Ticket y Bakery introducen mejoras al proporcionar equidad en la asignación de recursos y prevenir la inanición de procesos.

6- 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.

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



8-

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);

4- P(s1);

5- x = x - 2;
6- V(s2);
7- 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);

5- P(s2);
6- P(s1);
7- x = x * 3;
8- 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.

9- 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).

```
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]);
    }
}
```

10- 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.

11- 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)
        }
}
```