**1/9/2019**
**Programación Concurrente 2019**

Cuestionario guía - Clases Teóricas 3 a 6

1- Defina fairness. ¿Cuál es la relación con las políticas de scheduling? (NO DESCRIBA LOS DISTINTOS TIPOS DE POLITICAS DE SCHEDULING)
¿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?

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

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?

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).

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

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

7- OPCIONAL. Implemente una butterfly barrier para 8 procesos usando variables compartidas (puede consultar soluciones existentes en la web).

8- OPCIONAL. Una manera de ordenar *n* enteros es usar el algoritmo *odd/even exchange sort* (también llamado *odd/even transposition sort*). Asuma que hay *n* procesos P[1..*n*] y que *n* es par. En este algoritmo, cada proceso ejecuta una serie de rondas. En las rondas impares, los procesos con número impar P[impar] intercambian valores con P[impar+1] si los valores están desordenados. En las rondas con número par, los procesos con número par P[par] intercambian valores con P[par+1] si los valores están desordenados (en las rondas pares P[1] y P[*n*]
a) Determine cuántas rondas deben ejecutarse en el peor caso para ordenar los *n* números.
b) Escriba un algoritmo data parallel para ordenar un arreglo de enteros a[1:n] en forma ascendente
c) Modifique el algoritmo para terminar tan pronto como el arreglo fue ordenado
d) Modifique la respuesta de a) para usar *k* procesos; asuma que *n* es múltiplo de *k*
e) ¿Cómo afecta a su solución el hecho de poder ejecutar el algoritmo sobre un multiprocesador sincrónico?

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
```

10- Desarrolle 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).
   a) Utilizando semáforos
   b) Utilizando monitores

11-
a) Describa la técnica de *Passing the Baton*.
b) ¿Cuál es su utilidad en la resolución de problemas mediante semáforos?
c) ¿En qué consiste la técnica de *Passing the Condition* y cuál es su utilidad en la resolución de problemas con monitores?
d) ¿Qué relación encuentra entre *passing the condition* y *passing the baton*?

12- 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**

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

14- Modifique 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.
a) Resuelva con semáforos
b) Resuelva con monitores

15- 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.

16- (OPCIONAL:)
   (Broadcast atómico). Suponga que un proceso productor y n procesos consumidores comparten un buffer unitario. El productor deposita mensajes en el buffer y los consumidores los retiran. Cada mensaje depositado por el productor tiene que ser retirado por los n consumidores antes de que el productor pueda depositar otro mensaje en el buffer.
   a) Desarrolle una solución usando semáforos
   b) Suponga que el buffer tiene *b* slots. El productor puede depositar mensajes sólo en slots vacios y cada mensaje tiene que ser recibido por los *n* consumidores antes de que el slot pueda ser reusado. Además, cada consumidor debe recibir los mensajes en el orden en que fueron depositados (note que los distintos consumidores pueden recibir los mensajes en distintos momentos siempre que los reciban en orden). Extienda la respuesta dada en (a) para resolver este problema más general.
   c) resuelva a) y b) usando monitores.

17- Describa en qué consiste el problema de los “Drinking Philosophers”, y plantee al menos un esquema de solución con el mecanismo de sincronización que prefiera.

18- (OPCIONAL)

a) Implemente una butterfly barrier de *n* procesos usando semáforos (siendo *n* potencia de 2)

a) Utilice la barrera implementada en a) para resolver el algoritmo *odd/even exchange sort* para ordenar un arreglo de enteros a[1:n] en forma ascendente

b) Modifique la respuesta de a) para usar *k* procesos; asuma que *n* es múltiplo de *k*