1. (1 pto) Una de las siguientes afirmaciones NO es una característica de una condición de carrera (CC):

- El resultado de una operación sobre memoria compartida depende del orden de ejecución y velocidad relativa de dos o más hebras
- La CC sólo ocurre en sistemas multiprocesador o multicore
- La CC es un error potencial de concurrencia
- El resultado de una CC podría llevar a un resultado correcto

2. (2 pto) La sección crítica (SC) se define cómo:

- Código ejecutado por sólo una hebra realizando operaciones de escritura sobre memoria
- Código ejecutado por dos hebras realizando ambas operaciones de escritura sobre memoria local o privada
- Código ejecutado por dos o más hebras donde por lo menos una realiza operaciones de escritura sobre memoria global o compartida
- Código ejecutado por varias hebras donde todas realizan operaciones de escritura sobre memoria compartida

3. (2 ptos) Considere $i \neq j$ y $a \neq b$. Una solución incorrecta para proveer exclusión mutua (EM) permite que:

- Cuando la hebra h_i está en la SC_a, ninguna otra hebra h_j estará en la misma SC_a
- Si una hebra h_i ejecuta $enterSC_a()$ y ninguna otra hebra h_j se encuentra en la correspondiente SC_a, entonces h_i puede entrar a la SC_a
- Se garantiza que una hebra h_i que ejecuta $enterSC_a()$ eventualmente entrará a la SC_a.
- Cuando la hebra h_i está en la SC_a, otra hebra h_j podría estar en otra SC_b si ambas SC comparten la misma memoria

4. (2 pto) Identifique aquella afirmación que es verdadera:

- Toda instrucción en lenguaje de alto nivel es atómica
- Durante la ejecución de una instrucción de un lenguaje de alto nivel no ocurren cambios de contexto, pero pueden ocurrir entre varias instrucciones de *assembler*
- Aunque inanición implica deadlock, deadlock no implica inanición
- El código que ejecuta una hebra puede tener más de una SC

Escaneado con CamScanner

Considere el siguiente enunciado para las preguntas 5 y 6:

Un sistema computacional provee la instrucción de *hardware* fetch-and-add, instrucción que se comporta de la siguiente manera (C-style):

```
int fetch-and-add(int *ptr){
  int old = *ptr;
  *ptr = old + 1;
  return old;
}
```

Suponga que dos hebras ejecutan la función trabajador y que las variables `boleto` y `turno` son variables globales. Con la instrucción anterior se implementa el siguiente código con el objetivo de proveer EM.

```
void* trabajador(void *parametro){
    while(1){
      // código no crítico
      int mi_turno = fetch-and-add(&boleto);
      while(turno != mi_turno);
      SC();
      fetch-and-add(&turno);
      // código no crítico
    }
}
```

5. (1 pto) ¿Cuáles valores iniciales para las variables `boleto` y `turno` producen *deadlock*?

boleto = 1 y turno = 0
boleto = 0 y turno = 1
boleto = 1 y turno = 1
boleto = 0 y turno = 0

6. (1 pto) ¿Esta solución es libre de inanición?

Si, para los valores boleto = 1 y turno = 0
No, para los valores boleto = 0 y turno = 1
Si, para los valores boleto = 0 y turno = 0
No, para los valores boleto = 1 y turno = 1

Escaneado con CamScanner

Considere el siguiente enunciado para las preguntas 7, 8 y 9:

Por un túnel de una sola vía, sólo pueden pasar autos en una única dirección al mismo tiempo. Si pasa un auto en una dirección y hay autos en la misma dirección que quieren pasar, entonces tienen prioridad frente a los de la otra dirección (si hubiera alguno esperando para entrar al tunel). No hay restricción o límite con respecto al número de autos que puede haber en el tunel cruzando en una dirección.

El problema anterior se puede modelar de la siguiente manera utilziando semáforos contadores:

```
// Se asume que existen las variables globales contadorDirA y
// contadorDirB inicializadas en cero. También existen los semaforos
// semDirA, semDirB y puente binarios inicializados en uno. N hebras
// ejecutan autoDirA() y M hebras ejecutan autoDirB(), donde cada hebra
// simula un auto en una dirección (A o B).

void *autoDirA(void *data){

      semWait(&semDirA);
      contadorDirA++;
      if (contadorDirA == 1) semWait(&semTunel);
      semSignal(&semDirA);

      cruzarTunel();

      semWait(&semDirA);
      contadorDirA--;
      if (contadorDirA == 0) semSignal(&semTunel);
      semSignal(&semDirA);
}

void *autoDirB(void *data){

      semWait(&semDirB);
      contadorDirB++;
      if (contadorDirB == 1) semWait(&semTunel);
      semSignal(&semDirB);

      cruzarTunel();

      semWait(&semDirB);
      contadorDirB--;
      if (contadorDirB == 0) semSignal(&semTunel);
      semSignal(&semDirB);
}
```


Escaneado con CamScanner

7. (1 pto) Suponga que alguna ejecución concurrente ha producido estos valores donde la variable t representa valores discretos del tiempo. Para los valores de contadorDirA y contadorDirB se puede afirmar qué:

	t1	t2	t3	t4	t5	t6	t7	t8
contadorDirA	0	1	2	1	0	0	1	1
contadorDirB	0	0	0	1	1	2	1	0

La secuencia es inválida, dado que no es posible que en t4 contadorDirA y contadorDirB tomen el valor 1
La secuencia es inválida, dado que no es posible que en t7 contadorDirA y contadorDirB tomen el valor 1
La secuencia es inválida, por que los valores nunca superan en valor 2
La secuencia es válida

8. (2 pto) En el código se puede identificar la siguiente SC:

La sección crítica que accede a contadorDirB entre hebras que ejecutan autoDirB()
La sección crítica que accede a contadorDirB entre hebras que ejecutan autoDirA()
La sección crítica cruzarTunel() entre las hebras que ejecutan autoDirA()
La sección crítica cruzarTunel() entre las hebras que ejecutan autoDirB()

9. (1 pto) ¿Cuándo la solución siempre sufrirá inanición?

Cuando N y M son finitos
Cuando N y M son finitos y N es 10M
Cuando N y M son finitos y M es 10N
Cuando N y M tienden a infinito

10. (1 pto) Identifique aquella afirmación que NO es verdadera para deadlock:

Ocurre deadlock en un conjunto de hebras si algunas hebras están esperando un recurso que sólo puede liberar (o generar en caso de recursos consumibles) otro hebra del conjunto.
El algoritmo del banquero asegura que nunca ocurrirá deadlock
Para que un sistema pueda caer en deadlock, se deben cumplir 3 condiciones: exclusión mutua, hold and wait y no desapropiación.
Una solución que involucre recursos consumibles puede generar deadlock.

Escaneado con CamScanner

11. (1 pto) Para una solución de concurrencia basada en monitores, No es posible que:

- No cumpla con el requerimiento de proveer EM
- No cumpla con el requerimiento de no producir deadlock
- No cumpla con el requerimiento de no producir inanición
- No cumpla con el requerimiento de permitir progreso

Considere el siguiente enunciado para las preguntas 12 y 13:

Imagine una peluquería con un sólo peluquero, una sola silla de peluquería y una sala de espera con N sillas (N puede ser cero) para que esperen los clientes. Si no hay clientes, el peluquero se duerme en la silla de peluquero. Si hay por lo menos un cliente en la sala de espera, este debe despertar al peluquero. Si un cliente llega cuando el peluquero está trabajando (cortando el pelo), se sienta en una silla en la sala de espera si hay sillas disponibles, sino se va de la peluquería y vuelve en otro momento. Cuando el peluquero termina de cortar el pelo, verifica si hay clientes en la sala de espera para atenderlo, sino, vuelve a dormir en la silla de peluquero.

El problema anterior se puede modelar utilizando semáforos contadores de la siguiente manera:

```
semaphore peluqueroListo = 0;
semaphore salaDeEspera = 1;

semaphore hayClientes = 0;
int sillasLibres = N;

// Una hebra ejecuta la funcion peluquero
void *peluquero(void *data){
  while(1){
    semWait(&hayClientes);
    semWait(&salaDeEspera);
    sillasLibres++;
    semSignal(&peluqueroListo);
    semSignal(&salaDeEspera);
    // El peluquero le corta el pelo al cliente
  }
}

// Existen N hebras que ejecutaran la funcion cliente
void *cliente(void *data){
  while(1){
    semWait(&salaDeEspera);
    if (sillasLibres > 0) {
      sillasLibres--;
```


Escaneado con CamScanner

```
      semSignal(&hayClientes);
      semSignal(&salaDeEspera);
      semWait(&peluqueroListo);
      // El peluquero le corta el pelo al cliente
    } else semSignal(&salaDeEspera); // el cliente se va, no espera
  }
}
```

12. (1 ptos) Cuál de las siguientes afirmaciones es correcta:

- El peluquero nunca le cortará el pelo a un cliente dado que el semáforo hayClientes nunca es distinto de cero
- Un cliente nunca será atendido por el peluquero dado que el semáforo peluqueroListo nunca es distinto de cero
- Los clientes pueden sufrir inanición
- El semáforo salaDeEspera no es necesario, los demás semáforos proveen EM

13. (2 ptos) Cuál de las siguientes afirmaciones es incorrecta:

- Si dos clientes llegan al mismo tiempo, la solución registra que sólo un cliente decrementó el valor de hay~~Sillas~~ (osea, dos clientes se sentaron en la *misma silla*)
- Un mismo cliente puede ser atendido más de una vez
- Clientes que no estén en la sala de espera no serán atendidos
- La sección crítica corresponde al acceso a la variable compartida sillasLibres, *mientras* que el acto de cortar el pelo es coordinación entre hebras que se ejecutan en concurrencia

Escaneado con CamScanner