Universidad Nacional Autónoma de México
Facultad de Ciencias

Cómputo Concurrente

Semestre: 2025-1

Examen 1

García Ponce José Camilo 319210536

Parte Teórica

1. ¿Qué prefieres? ¿Comprar un uniprocesador que ejecute 5 millones de instrucciones por segundo, o un multiprocesador de 10 cores en donde cada core (procesador) ejecute 1 millón de operaciones por segundo? Utiliza la Ley de Amdahl para decidir la opción que más te conviene (más eficiente), si te dedicas al cómputo paralelo y casi todo tu trabajo es 84% paralelizable.

Sabemos que la fórmula de la ley de Amdahl para el speedup es $S = \frac{1}{(1-p)+\frac{p}{n}}$ donde p es la fracción paralelizable del programa y n es el número de procesadores. Ahora notemos que en el caso del uniprocesador solo tenemos un procesador por lo tanto no se pueden paralelizar las tareas/instrucciones, entonces en este caso se realizan 5 millones (5,000,000) de instrucciones por segundo.

Y para el caso de un multiprocesador de 10 cores tenemos que $p = \frac{84}{100}$ y $n = 10$ por lo tanto $S = \frac{1}{(1-\frac{84}{100})+\frac{\frac{84}{100}}{10}}$ lo cual es $S = \frac{250}{61} \approx 4.098$, por lo tanto tenemos un speedup de 4.098 veces, por lo tanto como cada procesador hace 1 millón de operaciones y vimos que el speedup es de 4.098, por lo tanto tenemos que el multiprocesador hace 4098000 tareas por segundo tomando en cuenta el porcentaje de trabajo que es paralelizable $(1000000 \times 4.098 = 4098000)$, por lo cual nos conviene más el uniprocesador que ejecuta 5 millones de instrucciones por segundo.

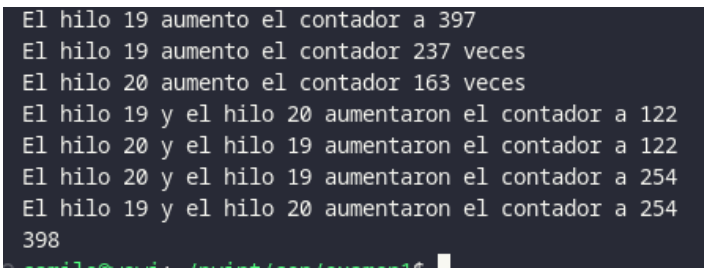
2. Eres uno de los P prisioneros arrestados. El guardia, un computólogo retirado, realiza el siguiente anuncio:
   Hoy se reunirán a planear una estrategia, sin embargo, después se les aislará en celdas y no podrán comunicarse con nadie.
   Preparé un cuarto con un interruptor que no está conectado a nada y que solo tiene dos estados: On/Off. A cada uno lo pasaré al cuarto de forma arbitraria, para cualquier número N de prisioneros, eventualmente visitarán el cuarto N veces. En cualquier momento alguno de ustedes declarará: "Ya hemos pasado todos" Si es correcto los liberaré, sino, serán alimento de cocodrilos.
   a. Crea una estrategia ganadora cuando no conoces el estado inicial del interruptor. *Comparte el pseudocódigo o una descripción*
      La estrategia es tener un prisionero que funcione como un contador. La estrategia para los demás prisioneros sera que si entran al cuarto y encuentran al interruptor en estado Off entonces lo pondrán en modo On, pero esta acción solo la podrán realizar dos veces, solo pondrán el interruptor en estado On las dos primeras veces que entren al cuarto y el interruptor esté en estado Off, la tercera y posterior veces que encuentren al interruptor en modo Off no van a realizar nada. Ahora la estrategia para el prisionero que será el contador, el tendrá un

contador que al inicio estará en 0, luego cada vez que entre al cuarto y el interruptor esté en estado On, lo va a poner en estado Off y aumentará su contador en 1, así sucesivamente hasta que su contador llegue al número 2n-2, cuando llegue a 2n-2 va a avisar que ya todos pasaron al cuarto, y si al entrar al cuarto encuentra el interruptor en Off no va a realizar ninguna acción.

b. Argumenta porque tu estrategia sí es correcta para cualquier número de N prisioneros.
   Esta solución sirve, ya que como cada prisionero, que no es el contador, solo va a subir el estado del interruptor a On, por lo tanto van a subir a On $2(n-1) = 2n-2$ veces, por lo tanto si el interruptor empieza en Off, el prisionero contador va a terminar luego de ponerlo en Off 2n-2 veces, es decir luego de que cada otro prisionero paso 2 veces al cuarto y por lo tal esta bien. Pero si el interruptor está On inicialmente, entonces tendremos que el prisionero contador va a terminar de contar luego de que los demás prisioneros cambien el estado del interruptor a On 2n-3 veces, ya que $2n-3+1 = 2n-2$ (el 1 viene del estado inicial del interruptor), por lo tanto sabemos que todos los prisioneros ya pasaron al menos una vez al cuarto, ya que si alguno de los otros prisioneros no habría pasado al cuarto entonces tendriamos que se cambio el estado a On 2n-4 veces por los prisioneros $2(n-2) = 2n-4$, y sumando el estado On inicial del interruptor tendríamos que el prisionero contador contó 2n-3 veces por lo tanto no podría haber terminado, ya que 2n-3 es menor a 2n-2, por lo tanto solo terminará cuando todos los prisioneros hayan pasado al menos una vez al cuarto. Ahora veamos que siempre terminara de contar el prisionero, ya que cada a lo más 2n visitas va a encontrar el interruptor en On (debido a que cada prisionero pasa al menos una vez cada N visitas), por lo tanto en a lo más $2n \times (2n-2) = 4n^4 - 4n$ visitas va a poder avisar que todos ya pasaron.
   De esta manera si tenemos que n es 2, entonces el prisionero contador se va a detener cuando su contador llegue a $2(2) - 2 = 2$, lo cual nos dice que el otro prisionero al menos paso una vez al cuarto con el interruptor, ya que si el interruptor inicia en On entonces el otro prisionero lo puso en On una vez y si inicia en Off entonces el otro prisionero lo puso en On dos veces, entonces si funciona bien.
   Supongamos que funciona para cuando es n es m, ahora veamos cuando n es m+1, entonces tendremos que el prisionero contador va a avisar que ya pasaron todos cuando su contador llegue a $2(m+1) - 2 = 2m$, por lo tanto sabemos que con m prisioneros se detuvo en $2(m) - 2 = 2m - 2$, y sabemos que cada uno de los

prisioneros que no son el contador pasaron (en general todos pasaron, pero nos importa que el contador contó que m-1 prisioneros pasaron), entonces tenemos que solo nos va a faltar que pase un prisionero y cómo nos detenemos cuando contador sea $2m$ entonces significa que nos falta contar dos veces por lo tanto al menos una de esas veces va a tener que ser el prisionero nuevo, ya que los demás ya pasaron. Pero veamos que si el interruptor empezó con estado Off entonces cuando se contó 2m-2 significa que todos los demás prisioneros ya pasaron 2 veces y por lo tanto las próximas dos veces que contemos serán del nuevo prisionero pero si el interruptor empezó con estado On entonces cuando se contó 2m-2 significa que todos los demás prisioneros ya pasaron 2 veces excepto un prisionero que solo pasó una vez, y por lo tanto las próximas dos veces que contemos tendremos que al menos una será del nuevo prisionero y de tal manera todos los prisioneros pasaron.

3. Considera la siguiente implementación de un candado para dos hilos *FakePeterson*.
    a. ¿Cumple con Exclusión mutua? Demuéstralo.
    No cumple con exclusión mutua, para esto veamos una ejecución, digamos que el hilo A hace la invocación del método lock() y un poco despues tambien el hilo B hace la invocación del método lock(), inicialmente los valores de flag y victim son false, entonces tendremos que A esta por entrar al while y revisa la condición de flag == true, a lo cual obtiene false y no entra al while, luego B está por entrar al while y revisa la condición de flag == true, a lo cual obtiene false y no entra al while (todo esto lo hace antes de que A actualice el valor de flag), después A realiza la línea flag = true, luego B realiza flag = true, posteriormente A esta por entrar al segundo while y revisa la condición de victim == true, a lo cual obtiene false y no entra al while, luego B está por entrar al segundo while y revisa la condición de victim == true, a lo cual obtiene false y no entra al while (todo esto lo hace antes de que A actualice el valor de victim), despues A realiza la lines victim = true y entra a la sección critica, y por ultimo B hace victim = true y entra a la sección critica. De esta manera tanto A y B entraron a la sección crítica al mismo tiempo. Además en la ejecución del programa se encontraron casos donde el contador era menor a 400.



b. Argumenta si cumple con Deadlock-free.
   Si cumple con Deadlock-free, esto debido a que ninguno de los dos hilos se puede quedar esperando a obtener el candado, si esto pasara entonces tendríamos que algun hilo se quedo en algun while atrapado, pero veamos que esto no puede pasar, ya que inicialmente los valores de flag y victim son false, por lo tanto no entran al while y solo se actualizan los valores a true luego de su while respectivo, además si tenemos que un while está en el while de flag, entonces significa que algún hilo ya pasó por ese while y puso flag en true, notemos que el hilo que puse flag en true no entro al while de vicitim ya que para que victim fuera true necesitamos que un hilo ya pasara el while de victim pero solo tenemos dos hilos, entonces estaría en la sección crítica, por lo tanto para que el primer hilo salga del while de flag sera hasta que flag sea false en el unlock del hilo en la seccion critica, de manera similar para cuando un hilo esta en el while de victim significa que un hilo esta en la seccion critica, ya que para tener vicitim como true se necesita que alguien ya saliera del while de victim, por lo tanto luego del unlock va a salir del while de victim. En resumen nos damos cuenta que ninguno de los dos hilos se puede quedar atrapado en un while, ya que para el while de flag tendríamos que tener a flag en true pero esto solo pasa si un hilo ya paso por este while y llegó a la línea 5 por lo tanto un hilo ya paso del while y los dos hilos no se pueden quedar en el while de flag al mismo tiempo, y para que alguien se quede en el while de victim necesitamos que victim sea true pero esto solo pasa si un hilo ya paso por este while y llegó a la línea 7 por lo tanto un hilo ya paso del while y los dos hilos no se pueden quedar en el while de victim al mismo tiempo. Por lo tanto no existe un caso donde los dos hilos se queden atrapados en un while y así tenemos que se cumple Deadlock-free, ya que si algunos hilos tratan de adquirir el candado entonces algún hilo lograra adquirir el candado, nunca se congela el sistema.

4. Otra generalización del algoritmo Peterson, además de Filter, consiste en asignar un candado de Peterson en cada nodo de un árbol binario. Supón que n es una potencia de dos, n corresponde al número de hilos. Dos hilos se asignan en una hoja del árbol (candado Peterson). Cada candado trata a un hilo como hilo 0 y hilo 1. En el método de adquirir (lock()) del árbol, el hilo debe adquirir todos los candados desde la hoja hasta la raíz. Cuando deja el candado (unlock()) debe dejar cada candado desde la raíz hasta las hojas. Considera que en cualquier momento un hilo puede detenerse por un periodo finito de tiempo (pueden tomar vacaciones, pero no detenerse por completo).
   a. Argumenta si cumple con Exclusión mutua.

Si cumple exclusión mutua, ya que para que no se cumpla exclusión mutua necesitamos que dos hilos entren a la sección crítica al mismo tiempo, entonces para que esto pase necesitamos que en el candado de la raíz lleguen dos hilos (que esto si puede pasar) y que ambos hilos ganen el candado pero esto esto último no puede pasar ya que sabemos que el candado Peterson cumple exclusión mutua por lo tanto solo un hilo puede entrar a la sección crítica y eso significa que solo un hilo puede ganar el candado del nodo raíz (en general en cada nodo sólo puede ganar el candado un hilo), es decir funciona como un torneo donde en cada nodo del árbol solo hay un ganador, por lo tanto se cumple exclusión mutua.

b. Argumenta si cumple con Starvation-free.
   Si cumple starvation-free, ya que para que no se cumpla starvation-free necesitamos que un hilo nunca logre obtener el candado y entrar a la sección crítica, y esto solo es posible si se queda atorado o congelado en algún nodo del árbol, pero para que esto pase significa que nunca puede obtener el cando de Peterson de ese nodo, lo cual no puede pasar ya que Peterson cumple Starvation-free por lo tanto vamos a saber que eventualmente cada hilo va a subir por los nodos del árbol hasta llegar a la raíz y poder entrar a su sección crítica. Por cual cumple starvation-free, todo gracias a que Peterson cumple starvation-free. Y además como cada hilo no se puede detener por completo en algún momento podrá entrar a su sección crítica.

c. ¿Existe una cota superior en el número de veces que un hilo puede ser aventajado? Argumenta por qué.
   Primero notemos que este candado no va a ser justo, esto debido a que si un hilo se queda dormido entonces los hilos que están subiendo por las otras ramas del árbol pueden obtener el candado y entonces un hilo que llegó a pedir el candado luego del hilo que se durmió lo puede obtener primero, y de esta manera no se cumpliria que fuera FIFO. De esta manera sabemos que un hilo puede ser aventajada varias veces, pero no podemos tener una cota superior, esto debido a que las veces que va a ser aventajado depende de la cantidad de tiempo en la que se quede dormido.

5. Argumenta si la siguiente ejecución es una ejecución de la Queue Wait-free para 2 hilos (recuerda que esta implementación es linealizable). Si es así muestra una linearización.

Sí es linealizable, para esto tendremos las siguientes historias (diremos que el objeto que se llama O).

- Tenemos que H es:
  q O.enq(1)

q O : true
p O.deq()
p O : 1
p O.deq()
p O : empty
q O.enq(2)
p O.deq()
p O : 2

- Después tenemos que la extensión de H o H' es:

q O.enq(1)
q O : true
p O.deq()
p O : 1
p O.deq()
p O : empty
q O.enq(2)
q O : true
p O.deq()
p O : 2

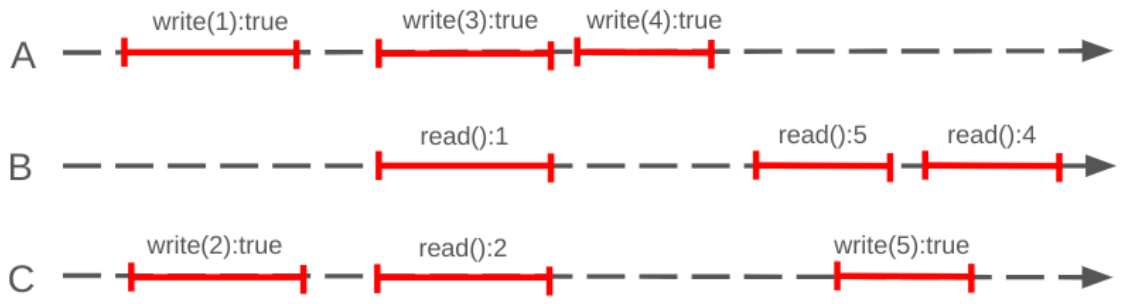
Solo agregamos la respuesta de q O.enq(2) como q O : true

- Por otro lado, para complete(H') será la misma H', ya que no tiene llamadas incompletas (sin respuesta)
- Y por último, la linealización S será complete(H') (entonces también es complete(H'))

Así tenemos que sí es linealizable.

6. Crea una ejecución de un registro que sea regular, que no sea linealizable y que no sea consistente en la inactividad.

Tendremos esta ejecución.



Ahora veamos que sí cumple lo que se pide.

Si cumple que sea un registro regular, ya que cuando un read() y write() se traslapan tenemos que el read() regresa un valor viejo o nuevo, y también las escrituras son atómicas, pero las lecturas no son atómicas.

Cumple con que no sea linealizable, ya que podemos observar la ejecución a partir del write(4):ok no se puede linealizar, esto debido a que tenemos el read():5 que termina antes del read():4, pero el write(4):ok termino antes de que el write(5):true empezará, entonces no es linealizable (se leyó un valor viejo).
Cumple con que no sea consistente en la inactividad, ya que podemos observar la ejecución a antes del write(4):ok no se es consistente en la inactividad, esto debido a que los write(1):true y write(2):true son separados de los read():1 y read():2 por un periodo de inactividad, por lo tanto no podemos acomodarlos de una manera para que tenga sentido.
Entonces si se cumple todo lo que se pide.

7. ¿La ejecución corresponde a la implementación del Snapshot Wait-free (figura 4)? Argumenta tu respuesta.

No corresponde el arreglo a_table a la ejecución dada, veamos la razón por la cual pasa esto.
Primero notemos el valor a_table[A].snap, tenemos que solo esta el valor a para el hilo A y puros valores por default para los demás hilos, entonces notemos que los valores de a_table sólo se modifican en los métodos update(), por lo tanto tenemos que a_table[A] solo se pudo modificar en el método update(a):ok, entonces veamos como funciona un update(), para esto primero se hace un scan() y en ese esta se van a hacer dos collects para ver los valores de los demás hilos, entonces aquí tenemos dos casos que no veamos ningún valor para todos los hilos o veamos el valor actualizado del hilo B (del update(b) si termina antes del scan() de update(a)), por lo tanto en ambos casos no vamos a ver al valor a, luego de realizar el scan(), notamos que los que se obtiene se usa para actualizar a_table[A] (línea 6), por lo tanto a_table[A].snap solo debería tener puros valores por default para todos los hilos o solo el valor b y los demás default (esto ya que un scan() se traslapa con un update()), pero esto no pasa en los valores del arreglo a_table, por lo tanto no coincide con la ejecución, si el hilo A hiciera otro update si podría tener ese snap en su posición de la a_table.
Después veamos que los valores de a_table[B].snap tampoco son válidos (por la misma razón que para el hilo A), tenemos que solo esta el valor a para el hilo B y puros valores por default para los demás hilos, entonces recordando que los valores de a_table sólo se modifican en los métodos update(), tenemos que a_table[B] solo se pudo modificar en el método update(b):ok, entonces veamos como funciona un update(), para esto primero se hace un scan() y en ese esta se van a hacer dos collects para ver los valores de los demás hilos, entonces aquí tenemos dos casos que no veamos ningún valor para todos los hilos o veamos el valor actualizado del hilo A (del update(a) si termina antes del scan() de update(b)), por lo tanto en ambos casos no vamos a ver al valor a, luego de realizar el scan(), notamos que los que se obtiene se usa para actualizar a_table[B] (línea 6), por lo tanto a_table[B].snap solo debería tener puros valores por default para todos los hilos o solo el valor a y los demás default (esto ya que un scan() se traslapa con un update()), pero esto no

pasa en los valores del arreglo a_table, por lo tanto no coincide con la ejecución, si el hilo A hiciera otro update si podría tener ese snap en su posición de la a_table.
Luego revisamos que los valores de a_table[D].snap tampoco son válidos, esto debido a que como hace un update (update(c)) solo una vez, entonces ahí es donde modificó su valor en a_table, pero ahora notemos que cuando realiza el scan() de este update() tenemos que no hay otro update() concurrente por lo tanto el scan() solo realiza dos collects y regresa los valore a, b y los demás como default, por lo tanto notamos que no puede ser al valor d (aun no se actualiza), por lo tanto en el snap de a_table[D] solo solo deben estar los valores a (para el hilo A), b (para el hilo B) y dos defaults (para los hilos C y D).
Y por último veamos los valores de a_table[C].snap, los cuales si son válidos debido a que el hilo C nunca hizo un update y por lo tanto nunca modificó su entrada en la a_table.
Por lo tanto tenemos que los valores snap de la a_table no corresponden a la ejecución de un Snapshot Wait-free.

8. En la implementación de un snapshot wait-free (Figura 4), si el arreglo de registros a table son registros regulares y no atómicos. Argumenta si el snapshot seguirá siendo linealizable o no.

Primero notemos las diferencias entre un registro regular y uno atómico, la principal diferencia es que los registros atómicos tienen lecturas atómicas mientras que los registros regulares no, por lo tanto tenemos que si usando un registro regular y dos operaciones de lectura (por el mismo hilo) se hacen al mismo tiempo que una operación de escritura, la primera operación de lectura puede regresar puede regresar el valor con el que se va a actualizar y la segunda lectura puede regresar el valor que existía antes de actualizar el valor (como en el ejercicio 6), por lo tanto generando algo que no se puede linealizar.
Por lo visto arriba tenemos que si en el snapshot se realiza una operación de scan() y update() al mismo tiempo (varios update() al mismo tiempo), vamos a tener que los valores que regresen los collect() pueden no ser consistentes o con los otros valores que regresen los collect() de otros métodos, lo cual nos causaría que existirían ejecuciones donde no sea linealizable, ya que las lecturas realizadas por los collect() no podemos garantizar que sean atómicas cuando se realiza un update de manera concurrente. Y de esta manera podríamos obtener que varios scan() de diferentes hilos regresen valores no consistentes o que tengan sentido.

Parte Práctica

1. Prueba el candado FakePeterson del ejercicio 3, utiliza un ExecutorService para ejecutar 400 tareas, considera que una tarea es un incremento a un contador sin consistencia.

El codigo se encuentra en los archivos FakePeterson.java, CounterNaive.java y Ejercicio.java

2. Si el candado cumple con exclusión mutua, siempre contará 400 tareas, vuelve a responder: ¿Cumple con exclusión mutua?

No cumple con exclusión mutua, ya que hay ocasiones donde no cuenta 400 tareas, por ejemplo la siguiente (notando que ambos hilos contaron la misma tarea)

```
El hilo 19 aumento el contador a 387
El hilo 20 aumento el contador a 388
El hilo 19 aumento el contador a 389
El hilo 20 aumento el contador a 390
El hilo 19 aumento el contador a 391
El hilo 19 aumento el contador a 392
El hilo 19 aumento el contador a 393
El hilo 19 aumento el contador a 394
El hilo 19 aumento el contador a 395
El hilo 19 aumento el contador a 396
El hilo 19 aumento el contador a 397
El hilo 19 aumento el contador a 398
El hilo 19 aumento el contador 154 veces
El hilo 20 aumento el contador 246 veces
El hilo 20 y el hilo 19 aumentaron el contador a 370
El hilo 19 y el hilo 20 aumentaron el contador a 370
399
```

3. ¿Cambia el resultado si utilizas volátiles en ambas variables o en ninguna variable? Argumenta por qué.

Con ambas variables no cambia el resultado, sigue sin cumplir exclusión mutua, esto ya que aunque ambas variables tengan volatile no van a poder evitar que ambos hilos eviten entrar a los whiles (como se explicó en el ejercicio 3 de la parte teórica), ya que puede pueden seguir revisando la condición de los whiles antes de que el otro hilo ponga las variables en true.

```
El hilo 19 aumento el contador a 387
El hilo 20 aumento el contador a 388
El hilo 19 aumento el contador a 389
El hilo 20 aumento el contador a 390
El hilo 20 aumento el contador a 391
El hilo 19 aumento el contador a 392
El hilo 20 aumento el contador a 393
El hilo 20 aumento el contador a 394
El hilo 19 aumento el contador a 395
El hilo 20 aumento el contador a 396
El hilo 20 aumento el contador a 397
El hilo 19 aumento el contador a 398
El hilo 19 aumento el contador 184 veces
El hilo 20 aumento el contador 216 veces
El hilo 19 y el hilo 20 aumentaron el contador a 326
El hilo 20 y el hilo 19 aumentaron el contador a 326
399
```

Con ninguna variables tampoco cambia el resultado, sigue sin cumplir exclusión mutua, esto ya que si ambas variables no tienen volatile van poder generar inconsistencias al intentar leer las variables y además no van a poder evitar que ambos hilos eviten entrar a los whiles (como se explicó en el ejercicio 3 de la parte teórica), ya que puede pueden seguir revisando la condición de los whiles antes de que el otro hilo ponga las variables en true.