**Programación Concurrente 2020**

Cuestionario guía - Clases Teóricas 1 y 2

**1- Mencione al menos 3 ejemplos donde pueda encontrarse concurrencia**

Navegador web, accediendo a distintas páginas al mismo tiempo, teléfono móvil, sistema de venta de pasajes.   
**2- Escriba una definición de concurrencia. Diferencie procesamiento secuencial, concurrente y paralelo.  
  
Concurrencia.** Es la capacidad de realizar múltiples actividades al mismo tiempo. Un programa concurrente especifica dos o más “programas secuenciales” que pueden ejecutarse concurrentemente en el tiempo como tareas o procesos.   
Un Programa concurrente va a estar formado por varios procesos, este programa es para resolver un único problema, ese programa va a estar compuesto por varios procesos, distintos comportamientos, pueden ser independientes los procesos cada uno resuelve su parte sin interactuar, pero vemos cuando estos procesos cooperan o compiten, entonces los procesos puede ser de tres formas, **independientes, cooperar o competir**

Procesamiento secuencial, un único flujo de control, ejecuta una instrucción y cuando finaliza la siguiente, monoprocesador. La solución secuencial nos fuerza a establecer un estricto **orden temporal**.   
  
Procesamiento concurrente, múltiples hilos de ejecución, un sóla máquina dedica una parte del tiempo a cada componente del objeto ⇒ Menor speedup.   
Si tuviera una sola máquina, aprovechar los tiempos muertos para ir avanzando en otro de los módulos, así vamos alternando el uso de la máquina entre las distintas piezas. Esto se llama **concurrencia sin paralelismo de Hardware.** Porque el procesamiento es concurrente pero la máquina es una sola. Se logra reducir el tiempo, al aprovechar los tiempos en que la máquina estaba ociosa. Ambos casos son concurrentes, con paralelismo o no.

Distribución de carga de trabajo. Necesidad de compartir recursos. Necesidad de esperar en puntos clave. Necesidad de comunicarse. Necesidad de recuperar el “estado” de cada proceso al retomarlo.   
  
Procesamiento paralelo, ejecución concurrente en múltiples procesadores con el objetivo principal de reducir el tiempo de ejecución, no hay dependencia entre cada procesador. Cada una puede trabajar al mismo tiempo en una parte, esto lleva menor tiempo para realizar el trabajo, pero es más costoso tener mas maquinas.

**3- Describa el concepto de deadlock y qué condiciones deben darse para que ocurra.**Dos (o más) procesos pueden entrar en deadlock, si por error de programación ambos se quedan esperando que el otro libere un recurso compartido. La ausencia de deadlock es una propiedad necesaria en los procesos concurrentes.  
  
Cuando un proceso necesita cierto recurso y no dejaron el anterior, ninguno puede continuar, van a estar esperando que el otro proceso libere un cierto recurso compartido que no lo deja hasta que no logre obtener el que necesita, entonces ninguno suelta su recurso hasta no obtener el otro y se produce deadlock(punto muerto), abrazo mortal porque ninguno proceso va a poder continuar.

**4 propiedades necesarias y suficientes para que exista deadlock son: La ausencia de deadlock es necesaria en los programas concurrentes (***yo debo asegurarme que alguna no se dé****)***

**Recursos reusables serialmente:** los procesos comparten recursos que pueden usar con exclusión mutua. Lo utiliza a medida que se va liberando pero con exclusión mutua.

**Adquisición incremental:** los procesos mantienen los recursos que poseen mientras esperan adquirir recursos adicionales. Recurso libre lo toma, otro libre lo toma, va tomando de a uno lo que necesita.

**No-preemption:** una vez que son adquiridos por un proceso, los recursos no pueden quitarse de manera forzada sino que sólo son liberados voluntariamente. Nadie lo puede forzar a dejarlo, una vez que terminó, no forzarlo.

**Espera cíclica:** Existe una cadena circular (ciclo) de procesos tal que cada uno tiene un recurso que su sucesor en el ciclo está esperando adquirir. Necesita un recurso compartido que tiene en siguiente proceso, a el de b, b el d c, c el de a.

**4- Defina inanición. Ejemplifique.  
Inanición →** alguno de los procesos nunca logre conseguir el recurso porque el otro gana, cuando más procesos hay es más probable que esto se dé.

Supongamos que los procesos son llamados por orden alfabético, es muy probable que los de la letra Z nunca lleguen a conseguir el recurso.

**5- ¿Qué entiende por no determinismo? ¿Cómo se aplica este concepto a la ejecución concurrente?**

No **determinismo**, en ejecución concurrente con la misma entrada se pueden obtener resultados diferentes *depende del orden en que se van ejecutando las cosas*.  
  
Es uno de los problemas asociados a la programación concurrente, hay un no determinismo implícito en el interleaving de procesos concurrentes. Esto significa que dos ejecuciones del mismo programa no necesariamente son idénticas ⇒ dificultad para la interpretación y debug, para debuggear algo es muy difícil porque uno quiere volver a ejecutar, para ver el error y esto puede haber cambiado.

**6- Defina comunicación. Explique los mecanismos de comunicación que conozca.**Comunicación entre procesos,es el modo en que se organizan y transmiten los datos, requiere especificar ciertos protocolos, como progresar y que se haga de forma correcta.  
  
 • **Por Memoria Compartida:** Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella. Necesito algún lugar de memoria donde todos los procesos puedan acceder, dejando los datos en alguna dirección donde otro pueda tomarlo. A través de esa memoria compartida, una físicamente y además los procesos acceder al mismo espacio de direcciones. Tengo que tener cuidado con el uso dos procesos quieren modificar la misma variable, necesito que la herramienta me brinde algún bloqueo o liberar el acceso a una posición de memoria. Lo que se usa semáforo, nos habilitará o no el paso. Una máquina que tenga una memoria compartida, que permita que los procesos puedan acceder al mismo espacio de direcciones.  
  
• **Por Pasaje de Mensajes.** Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos.   
También el lenguaje debe proveer un protocolo adecuado.   
Para que la comunicación sea efectiva los procesos deben “saber” cuándo tienen mensajes para leer y cuándo deben transmitir mensajes.  
Canales lógicos o físicos que me permiten transmitir, tengo que poder definir esos canales, sentencias primitivas para los datos que se envían a través de esos canales, a quien le envío, de quien recibo información, importante, a pesar de que está pensado a computadoras separadas conectadas a través de una red de interconexiones los procesos también podrían estar en una monoprocesador o multicore, canal lógico en vez de red.  
  
**7- a) Defina sincronización. Explique los mecanismos de sincronización que conozca.**

La sincronización es la posesión de información acerca de otro proceso para

coordinar actividades. Los procesos se sincronizan:

• **Por exclusión mutua**. Asegurar que sólo un proceso tenga acceso a un recurso compartido en un instante de tiempo solo por un proceso a la vez, y hasta que no termine no cambia. El uso de un recurso compartido, usado por un único proceso a la vez.   
Ejemplo: Una variable compartida que tiene que ser modificada, el uso de una impresora.

Si el programa tiene secciones críticas que pueden compartir más de un proceso, la exclusión mutua evita que dos o más procesos puedan encontrarse en la misma sección crítica al mismo tiempo.

**• Por condición.** Permite bloquear la ejecución de un proceso hasta que se cumpla una condición dada.

Ejemplo: elemento que hay en una cola, no puedo ir y sacarlo espero que este vacia para recién hacer el pop y sacar el elemento.

**b) ¿En un programa concurrente pueden estar presentes más de un mecanismo de sincronización? En caso afirmativo, ejemplifique***Ejemplo de los dos mecanismos de sincronización en un problema de utilización de un área de memoria compartida (buffer limitado con productores y consumidores).*

Problema concurrente que debe utilizar ambos, la utilización de memoria compartida, al cual todos los procesos pueden acceder y es limitada procesos productores y consumidores, no puedo dejar elementos hasta que alguien lo haya dejado, los consumidores tampoco pueden tomarlo hasta que al menos haya un dato en el buffer, se dan ambos tipos de sincronización, el consumidor se bloquea hasta que hay algo y el consumidor se bloquea hasta que haya espacio para dejar un elemento, esto es por condición.  
 Si hay más de un productor, los productores van a tener que sincronizar entre ellos por exclusión mutua para que de los productores uno solo pueda acceder.

Del otro lado tenemos la sincronización por exclusión mutua, si hay más de un productor para dejar un elemento pero un único elemento, por exclusión mutua van a tener que sincronizarse.

**8- ¿Qué significa el problema de “interferencia” en programación concurrente? ¿Cómo puede evitarse?**Interferencia: un proceso toma una acción que invalida las suposiciones hechas por otro proceso. Un proceso toma el estado y resulta que hay otros procesos que modificaron ese estado.

Ejemplo 1: variables compartidas, evitar la div por cero, chequea que sea distinto de 0, podría ocurrir que el proc 2 chequea y se da cuenta que es distinto de cero, pero antes modifica hacer la división el proceso a1 lo convierte en 0 y recien despues hace la división, generando un error, A1 invalida a A2 por interferencia que acá produce un error del programa.

Ejemplo 2 Si no utilizo excl mutua para trabajar con la variable, podría pasar q en algun momento ambos a la e están intentando incrementar, los dos quieren incrementar y se haría solo una vez el incremento, deberá haber incrementado dos, uno por cada uno, la interferencia acá yo trabajo con el valor de público sin embargo otro proceso lo modifico y me dej a mi trabajando con otro proceso.

**9- ¿En qué consiste la propiedad de “A lo sumo una vez” y qué efecto tiene sobre las sentencias de un programa concurrente? De ejemplos de sentencias que cumplan y de sentencias que no cumplan con ASV.**

Si yo tengo una expresión e , en un proceso, no ref a una vari ref a otra referenciada por otro proceso,  
• Si una expresión e en un proceso no referencia una variable alterada por otro proceso, la evaluación será atómica, aunque requiera ejecutar varias acciones atómicas de grano fino.   
• Si una asignación x = e en un proceso no referencia ninguna variable alterada por otro proceso, la ejecución de la asignación será atómica.  
  
 que involucra muchas acc atom, pero esas son solo sobre variables propias del proceso, ea acc, la puedo ver como atómica aunque involucre mas de una acción atómica de grano fino porque no estpa afectando al resto de las variables, una sit en la que cada y no de los procesos accede y modifica a cada uno de los procesos,   
  
por mas de que e impliqe modificar a un monotn de variables si es del propio proceso no van a afectar su ejecucion.   
**Prog concu donde c u de los proc tiene sus prop variables y no afecta la ejec de las variables de otros procesos, prog concu con proc disjuntos, esta sit no es lo mas comun en concurrencia, por lo tanto necesito establecer un reuqe mas debil que que los proc no compartan variables.   
Ref critica a una ref a una vari que es modificada por otro proceso, va a ser una ref a una var uqe modif otro de los procesos, critico y una sentencia de asig, de la forma x0 exp**

**safitace la prop de a lo sumo una vez, dos posib:**

**la exp contiene a la sumo una unica ref critica(una var ref por otro proc) y x no es ref por otro proceso,**

2) e no contiene referencias críticas, en cuyo caso x puede ser leída por otro proceso.   
  
Una sentencia de asignación x = e satisface la propiedad de “A lo sumo una vez” si:   
1) e contiene a lo sumo una referencia crítica y x no es referenciada por otro proceso, o  
 2) e no contiene referencias críticas, en cuyo caso x puede ser leída por otro proceso. Una expresiones e que no está en una sentencia de asignación satisface la propiedad de “A lo sumo una vez” si no contiene más de una referencia crítica.

A lo sumo una variable compartida y esta puede ser ref a lo sumo una vez.

La ejecución va a parecer atómica aunque no lo sea porque la variable compartida va a ser leída/escrita una sola vez.  
  
int x=0, y=0; No hay ref. críticas en ningún proceso.

co x=x+1 // y=y+1 oc; En todas las historias x = 1 e y = 1

//x ni y son modi por otro proceso!!! cumplen con la prop asv.   
  
int x = 0, y = 0; El 1er proceso tiene 1 ref. crítica. El 2do ninguna.

co x=y+1 // y=y+1 oc; Siempre y = 1 y x = 1 o 2

es modif por otro proceso pero no referenciada.

pero y no es ref por otro proceso.

int x = 0, y = 0; Ninguna asignación satisface ASV.

co x=y+1 // y=x+1 oc; Posibles resultados: x= 1 e y =2 / x = 2 e y = 1

Nunca debería ocurrir x = 1 e y = 1 → ERROR  
  
 y es una ref critica, modif en otro proce y además x estaa ref en otro proceso. NO CUMPLE

**10- Dado el siguiente programa concurrente:**

**x = 2; y = 4; z = 3;**

**co**

**x = y - z // z = x \* 2 // y = y - 1**

**oc**

**a) ¿Cuáles de las asignaciones dentro de la sentencia co cumplen con ASV?. Justifique claramente.  
x=y-z es modif por otro proceso y ademas desp referenciada.**

**z=x\*2 es modif y ref por otro.   
y=y-1 cumple.**

**b) Indique los resultados posibles de la ejecución**

**Nota 1: las instrucciones NO SON atómicas.**

**Nota 2: no es necesario que liste TODOS los resultados, pero si los que sean representativos de las diferentes situaciones que pueden darse.**

**11- Defina acciones atómicas condicionales e incondicionales. Ejemplifique.**await s < 0 y dec atomic , Await general: 〈await (s>0) s=s-1;〉 → acc atom condicional.  
o el await solo para e.m 〈x = x + 1; y = y + 1〉 → acciones atomicas incondicionales  
estado en el uqe uno fue incrementado y otro no. no visible para el resto.   
  
Caso particular, Ejemplo await para sincronización por condición:   
*Si B satisface ASV(sino no) porque podria pasar uqe asuma que la cond es falsa cuando por alguna interf se volvio verdadera* , puede implementarse como **busywaiting o spin loop do**  
si la cond satif la prop de als, la puedo implementar como busy waiting.  
  
〈await (count > 0) 〉(not B) → skip od (while (not B);)

**12- Defina propiedad de seguridad y propiedad de vida.**Una propiedad de un programa concurrente es un atributo verdadero en cualquiera de las historias de ejecución del mismo

Toda propiedad puede ser formulada en términos de dos clases: seguridad y vida.

**• seguridad (safety)**

Nada malo le ocurre a un proceso: asegura estados consistentes.

Una falla de seguridad indica que algo anda mal.

Ejemplos de propiedades de seguridad: exclusión mutua, ausencia de interferencia entre procesos, partial correctness.

**• vida (liveness)**

Eventualmente ocurre algo bueno con una actividad: progresa, no hay deadlocks.

Una falla de vida indica que las cosas dejan de ejecutar.

Ejemplos de vida: terminación, asegurar que un pedido de servicio será atendido, que un mensaje llega a destino, que un proceso eventualmente alcanzará su SC, etc ⇒ dependen de las políticas de scheduling.

**13- ¿Qué es una política de scheduling? Relacione con fairness. ¿Qué tipos de fairnes conoce?**La política de scheduling de las elegibles dice cual es la que va a ejecutar, fairnees(justicia) trata de garantizar de que todos los procesos puedan avanzar sin importar que haga el resto.  
Una acción atómica en un proceso es **elegible** si es la próxima acción atómica en el proceso que será ejecutada. Si hay varios procesos ⇒ hay varias acciones atómicas elegibles. Una política de scheduling determina cuál será la próxima en ejecutarse.  
La política de scheduling depende del so no del lenguaje.   
  
Fairness Débil. Una política de scheduling es débilmente fair si : (1) Es incondicionalmente fair y (2) Toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada, asumiendo que su condición se vuelve true y permanece true hasta que es vista por el proceso que ejecuta la acción atómica condicional.  
  
Si tengo una acción atom, donde await cond accion, si la cond se vuelve verd y se mantiene hasta q la ve el proces, si ocurre es debilemtne fair, esto solo no alcanza, porque puede pasar q la cond cambie, ademas todas acc atom elegible.

**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?**

El Fairness (justicia/equidad) es una característica de las políticas de scheduling, que indica si los procesos tendrán la oportunidad de avanzar, sin importar lo que hagan los demás. Existen distintos tipos: incondicionalmente-fair, débilmente-fair y fuertemente-fair.

Se aplica el concepto de fairness al scheduling que se haga de la base de datos como recurso compartido:

1. Si el ingreso de un lector permite el ingreso de otros, aún si hay escritores esperando → débilmente fair: el ingreso permanente de lectores causaría inanición a los escritores.
2. Si la llegada de un escritor impide el acceso de nuevos procesos y es atendida en primer lugar al liberarse la BD → débilmente fair: el ingreso constante de escritores causaría inanición a los lectores.
3. Si los procesos son atendidos en orden FIFO con acceso concurrente de los lectores → fair: todos serán atendidos.

La última opción, aunque es fuertemente fair, produce un delay innecesario en los lectores que lleguen tras un escritor que espera. Sin embargo, dada la naturaleza del problema, de evitar dicho delay se caería en un esquema de prioridad con riesgo de inanición a los escritores.

-Se puede aplicar el concepto de fairness a la política de scheduling que se utilice para

acceder a dicha BD.

Si el acceso de un lector permite el acceso de otros, aunque haya escritores esperando →

debilmente fair,​ un ingreso permanente de lectores podría causar inanición en los

escritores.

Si la llegada de un escritor impide el ingreso de otros y es atendida en primer lugar al

liberarse la BD → debilmente fair,​ el ingreso constante de escritores puede causar

inanición de lectores.

Si los procesos son atendidos en orden FIFO con acceso concurrente de lectores → fair,​ ya

que todos serán atendidos.

Esta última opción podría producir una demora innecesaria en los lectores que lleguen tras

un escritor que espera, pero para evitar esto se caería en un esquema de prioridad con

riesgo de inanición de escritores.

**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?**

* Recursos reusables serialmente: los procesos comparten recursos que pueden usar con exclusión mutua.
* Adquisición incremental: los procesos mantienen los recursos que poseen mientras esperan adquirir recursos adicionales.
* No-preemption: una vez que son adquiridos por un proceso, los recursos no pueden quitarse de manera forzada sino que sólo son liberados voluntariamente.
* Espera cíclica: existe una cadena circular (ciclo) de procesos tal que cada uno tiene un recurso que su sucesor en el ciclo está esperando adquirir.

**¿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- Analice las soluciones para las barreras de sincronización desde el punto de vista de la complejidad de la programación**

**y de la performance.**

**8- (OPCIONAL). Implemente una butterfly barrier para 8 procesos usando variables compartidas.**

**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 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).**

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

**12- 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.**

**13- (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.**

**14- (OPCIONAL) Implemente una butterfly barrier de n procesos usando semáforos (siendo n potencia de 2)**