**CUESTIONARIO CLASE 1 Y 2**

1. **Mencione al menos 3 ejemplos donde pueda encontrarse concurrencia.**
2. Videojuego de estrategia. Mientras mantiene actualizado el mapa en pantalla, calcula el movimiento de enemigos, captura órdenes de los usuarios, controla el desarrollo del juego, etc.
3. Reproductor de DVD. Mientras lee el dispositivo, decodifica audio y video, controla la temporización de subtítulos, envía las señales de salida, etc.
4. Procesador de texto. Mientras recibe entras del usuario, controla ortografía, realiza sugerencias, guardados automáticos, etc.
5. **Escriba una definición de concurrencia. Diferencie procesamiento secuencial, concurrente y paralelo.**

La concurrencia es ejecutar dos o más procesos en forma simultánea.

* Procesamiento Secuencial: cuando se tiene un único hilo de ejecución o control.
* Procesamiento Concurrente: cuando se tiene más un hilo de ejecución o control, no implicando la existencia de más de un procesador.
* Procesamiento Paralelo: procesamiento concurrente sobre más de un procesador, ejecutando cada uno un hilo diferente en un momento dado.

1. **Describa el concepto de deadlock y qué condiciones deben darse para que ocurra.**

Deadlock es un bloqueo permanente de un conjunto de procesos producto de la espera por recursos de uso exclusivo, cada proceso esperando algún recurso sin liberar a los que posee bajo control. Puede producirse entre dos procesos o entre varios, en forma circular.

Las condiciones necesarias para que se produzca (condiciones de Coffman) son:

* Exclusión mutua: existencia de al menos un recurso compartido por los procesos al cual solo puede acceder uno en un momento dado.
* Posesión y espera: al menos un proceso Pi ha adquirido un recurso Ri, y lo mantiene mientras espera al menos un recurso Rj que ya ha sido asignado a otro proceso.
* No expropiación: un recurso asignado a un proceso Pi no puede ser expropiado por otro proceso Pj.
* Espera circular: dado un conjunto de n procesos Pi, cada proceso retiene un recurso que espera su sucesor, esperando P1 el recurso retenido por Pn.

1. **Defina inanición. Ejemplifique.**

Inanición significa que un proceso no puede finalizar su ejecución ya que requiere de un recurso que nunca le es asignado.

El deadlock es un caso especial de la inanición, como así también el livelock que es una condición similar, pero en la que el estado de los procesos continúa cambiando, cada uno en respuesta a los cambios del otro, sin retomar el procesamiento.

Ejemplo de inanición: cualquier esquema de atención por prioridades. Una técnica común para evitarlo es el esquema de envejecimiento.

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

El no determinismo es la producción de resultados impredecibles: incluso ante los mismos valores y condiciones iniciales, puede obtener distintos resultados.

En un programa concurrente compuesto por un conjunto de n procesos, en un momento dado habrá n operaciones atómicas elegibles (asumiendo que todos están en estado ready). Así, pueden obtenerse distintas historias de la ejecución del programa aun en corridas con iguales valores de entrada. La historia especifica de una corrida será decidida NO por el programa concurrente, sino por la política de scheduling que aplique el SO.

1. **Defina comunicación. Explique los mecanismos de comunicación que conozca.**

Indica el modo en que se organizan y trasmiten datos entre tareas concurrentes. Esta organización requiere especificar ciertos protocolos que nos permitan controlar el progreso y la corrección.

Hay comunicación:

* Por Memoria Compartida: los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella.
* Por Pasaje de Mensajes: es necesario establecer un canal (lógico o físico) para transmitir información entre procesos. Para que la comunicación sea efectiva los procesos deben saber cuándo tienen mensajes para leer y cuando deben trasmitir mensajes.

1. **a) Defina sincronización. Explique los mecanismos de sincronización que conozca.**

La sincronización es la interacción entre procesos que permite controlar el orden en que se ejecutan, restringiendo las historias posibles solo a las deseables. Hay dos formas básicas:

- Por Exclusión Mutua: asegura que las sentencias en las secciones críticas de distintos procesos no puedan ejecutarse al mismo tiempo.

- Por Condición: retrasa la ejecución de un proceso hasta que una condición dada se vuelva verdadera.

**b) ¿En un programa concurrente pueden estar presentes más de un mecanismo de sincronización? En caso afirmativo, ejemplifique**

Ambos métodos de sincronización pueden estar presentes en un mismo programa ya que puede haber más de una circunstancia en la que los procesos precisen sincronizarse. Además, tiene distintos fines: delimitar secciones críticas y garantizar que el estado del programa sea adecuado antes de continuar la ejecución de un proceso.

Por ejemplo, puede lograrse la exclusión mutua utilizando sincronización por condición para los protocolos de entrada a las secciones críticas.

1. **¿Qué significa el problema de “interferencia” en programación concurrente? ¿Cómo puede evitarse?**

La interferencia es el resultado de dos o más procesos leyendo y escribiendo variables compartidas en un orden impredecible y, por tanto, con resultados impredecibles. En general, una instrucción atómica en un proceso interfiere con una precondición de una sección crítica o una suposición hecha por otro, si modifica su valor de verdad.

Puede evitarse por medio de:

* Variables disjuntas: validar que el conjunto de escritura de un proceso y el conjunto de referencias de otro sean distintos.
* Asunciones débiles: escribir los procesos considerando los efectos de otros procesos en ejecución, realizando menos asunciones o más generales.
* Invariante global: si todas las asunciones de los procesos pueden escribir de la forma I ^ L, donde I es un invariante global y donde L solo referencia variables locales o globales que solo ese proceso escribe, entonces los procesos están libres de interferencia.
* Sincronización: evitar interferencia mediante exclusión mutua o sincronización por condición.

1. **¿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.**

La propiedad de A lo sumo una vez indica que una expresión no causara interferencia con otros procesos si tiene como máximo una referencia critica. En el caso de una asignación variable = expresión, es necesario que ningún proceso lea la variable o que en ese caso, no haya referencias críticas en expresión.

Las sentencias de un programa concurrente pueden referenciar a lo sumo una variable compartida, ASV, para estar libres de interferencias. Si así lo hacen, su ejecución parecerá atómica al resto de los procesos.

|  |  |
| --- | --- |
| *No Cumplen ASV:*  int a = 3, j = 6  co  **j = a + j #2 ref. Críticas**  // j = 1 #1 ref. críticas  oc | *Cumplen ASV:*  int a = 3, j = 6  co     a = a + j #1 ref. Crítica, *a* NO es leída  // j = 1 #0 ref. críticas  oc |

1. **Dado el siguiente programa concurrente:**

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

**co**

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

**oc**

1. **¿Cuáles de las asignaciones dentro de la sentencia co cumplen con ASV? Justifique claramente.**

x = y – z 🡪 NO CUMPLE. Dos referencias críticas, y x es leída por otros procesos.

z = x \* 2 🡪 NO CUMPLE. Una referencia critica, y z es leída por otros procesos.

Y = y – 1 🡪 CUMPLE. No posee referencias críticas al lado derecho; y puede ser leída por otros procesos.

1. **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.**

Se asigna x, luego z y ultimo y: x=1, z=4, y=3.

Si los procesos ejecutan una instrucción atómica de bajo nivel por vez: x=1, z=4, y=3.

Se asigna x, luego y, ultimo z: x=1, z=4, y=3.

Se asigna y, luego z, ultimo x: x=-1, z=4, y=3.

Se asigna y, luego x, ultimo z: x=0, z=0, y=3.

Se asigna z, luego x, último y: x=0, z=4, y=3.

Se asigna z, luego y, ultimo x: x=-1, z=4, y=3.

Nótese que el valor de y para todas las ejecuciones presentadas se mantiene en 3, dado que su asignación es la única que cumple ASV. Las variables x y z varían en su resultado.

1. **Defina acciones atómicas condicionales e incondicionales. Ejemplifique.**

**FALTA COMPLETAR**

Incondicionales: son las acciones que se utilizan únicamente para exclusión mutua, es decir, aquellas que no llevan una condición booleana.

Condicionales: son acciones que si tienen la condición booleana de espera.

1. **Defina propiedad de seguridad y propiedad de vida.**

Propiedad de seguridad (safety): asegura estados consistentes (nada malo le ocurre a un proceso). Una falla de seguridad indica que algo anda mal.

Propiedad de vida (liveness): hay progreso, no hay deadlocks (eventualmente ocurre algo bueno con una actividad). Una falla de vida indica que las cosas dejan de ejecutar.

1. **¿Qué es una política de scheduling? Relacione con fairness. ¿Qué tipos de fairness conoce?**

Una política de scheduling se encarga de elegir la tarea siguiente que hay que admitir en el sistema y el proceso siguiente que hay que ejecutar.

Fairness: trata de garantizar que los procesos tengan chance de avanzar, sin importar lo que hagan los demás.

Tipos de Fairness:

* Fairness Incondicional: una política de scheduling es incondicionalmente fair si toda acción atómica incondicional que es elegible eventualmente es ejecutada.
* Fairness Débil: una política de scheduling es débilmente fair si:
  + Es incondicionalmente fair y si toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada, asumiendo que su condición se vuelve true y permanece hasta que es vista por el proceso que ejecuta la acción atómica condicional.
* Fairness Fuerte: una política de scheduling es fuertemente fair si:
  + Es incondicionalmente fair y si toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada pues su guarda se convierte en true con infinita frecuencia.