PROGRAMACION CONCURRENTE - EXAMEN FINAL 3-3-2010

En todos los casos, responda con claridad y sintéticamente.
En los casos que corresponda, NO SE CONSIDERARAN RESPUESTAN SIN JUSTIFICAR.
Es condición necesaria para la aprobación que el ejercicio 10 esté bien resuelto.
Tiempo Máximo 2 hs 15 min.

1. a) Defina programa concurrente, programa paralelo y programa distribuido.

Programa concurrente: Especifica dos o más programas secuenciales que pueden ejecutarse en forma concurrente, es decir en simultáneo, como tareas o procesos. Dichos programas pueden o no correr en múltiples procesadores.

Programa distribuido: Es un programa concurrente (o paralelo) en el cual los procesos se comunican por pasaje de mensajes. Es un caso de concurrencia con múltiples procesadores y sin memoria compartida.

Programa paralelo: Se asocia con la ejecución concurrente en múltiples procesadores que pueden tener memoria compartida, generalmente con el objetivo de incrementar performance.

b) Compare la comunicación por mensajes sincrónicos y asincrónicos en cuanto al grado de concurrencia y la posibilidad de entrar en deadlock.

En cuanto a la facilidad de la programación no varía mucho, lo que es importante en este aspecto es que si programamos con mensajes sincrónicos, el emisor del mensaje se va a quedar bloqueado hasta cuando el receptor lo reciba, por lo que hay que saber manejar de la manera más óptima las situaciones de comunicación para no generar demoras innecesarias, en el caso del uso de mensaje asincrónicos, el emisor del mensaje deposita el mensaje y posteriormente sigue su curso normal, no hay retraso.

En términos concurrente, el grado de concurrencia se reduce respecto de la sincronización por Pasaje de Mensaje Asincrónico, ya que siempre un proceso se bloquea. Pero en términos de memoria, utiliza menos memoria, debido a que la cola de mensajes asociada a un canal se reduce a 1 mensaje.

c) Defina el concepto de "continuidad conversacional" entre procesos

Es un modo de interacción entre clientes y servidores en File Servers, en los cuales contamos con un canal abrir compartido por cualquier FS, donde cada canal puede tener un solo receptor; por lo tanto para mantener una continuidad conversacional necesitamos un alocador de archivos separado que reciba pedidos de abrir y aloque un FS libre a un cliente. Cuando estos son liberados se debe avisar de dicho acontecimiento al alocador.

2. Cómo pueden clasificarse las arquitecturas multiprocesador:
a) según el mecanismo de control? Describa.

Esta clasificación se basa en la manera en que las instrucciones son ejecutadas sobre los datos.
Existen 4 clases:

I. SISD (Single Instruction Single Data): Instrucciones ejecutadas en secuencia, una por ciclo de instrucción. La memoria afectada es usada sólo por esta instrucción. Usada por la mayoría de los uniprocesadores. La CPU ejecuta instrucciones (decodificadas por la UC) sobre los datos. La memoria recibe y almacena datos en las escrituras, y brinda datos en las lecturas. Ejecución determinística.
II. SIMD (Single Instruction Multiple Data): Conjunto de procesadores idénticos, con sus memorias, que ejecutan la misma instrucción sobre distintos datos. El host hace broadcast de la instrucción. Ejecución sincrónica y determinística. Pueden deshabilitarse y habilitarse selectivamente procesadores para que ejecuten o no instrucciones. Los procesadores en general son muy simples. Adecuados para aplicaciones con alto grado de

regularidad, (por ejemplo procesamiento de imágenes). Ejemplos de esta arquitectura: Array Processors, CM-2, Maspar MP-1 y 2, Illiac IV.

III. MISD (Multiple Instruction Single Data): Los procesadores ejecutan un flujo de instrucciones distinto pero comparten datos comunes. Operación sincrónica (en lockstep). No son máquinas de propósito general ("hipotéticas", Duncan) Ejemplos posibles: múltiples filtros de frecuencia operando sobre una única señal, múltiples algoritmos de criptografía intentando crackear un único mensaje codificado.

IV. MIMD (Multiple Instruction Multiple Data): Cada procesador tiene su propio flujo de instrucciones y de datos. Cada uno ejecuta su propio programa. Pueden ser con memoria compartida o distribuida. Ejemplos: nCube 2, iPSC, CM-5, Paragon XP/S, máquinas DataFlow, red de transputers. Sub-clasificación de MIMD:

a) **MPMD (multiple program multiple data):** cada procesador ejecuta su propio programa (ejemplo con PVM).
b) **SPMD (single program multiple data):** hay un único programa fuente y cada procesador ejecuta su copia independientemente (ejemplo con MPI).

b) según la organización del espacio de direcciones? Describa.

I. Multiprocesadores de memoria compartida: La interacción se da modificando datos almacenados en la MC. Puede haber problema de consistencia. Existen dos esquemas:

a) **Esquemas UMA** con bus o crossbar switch (SMP, multiprocesadores simétricos).
b) **Esquemas NUMA** para mayor número de procesadores distribuidos.

II. Multiprocesadores con memoria distribuida: Procesadores conectados por una red. Cada uno tiene memoria local y la interacción es sólo por pasaje de mensajes. No hay problemas de consistencia. Grado de acoplamiento de los procesadores:

a) **Multicomputadores (tightly coupled machine):** Procesadores y red físicamente cerca. Pocas aplicaciones a la vez, cada una usando unconjunto de procesadores: Alto ancho de banda y velocidad.
b) **Redes (loosely coupled multiprocessor).**
c) **NOWs/Clusters.**
d) **Memoria compartida distribuida.**

c) según la red de interconexión? Describa.

Tanto las máquinas de MC como de MP pueden construirse conectando procesadores y memorias usando diversas redes de interconexión, estáticas y dinámicas. Las **redes estáticas** constan de links punto a punto. Típicamente se usan para máquinas de MP. Las **redes dinámicas** están construidas usando switches y enlaces de comunicación. Normalmente para máquinas de MC.
El diseño de la red de interconexión depende de una serie de factores (ancho de banda, tiempo de startup, paths estáticos o dinámicos, operación sincrónica o asincrónica, topología, etc.)

3. Suponga los siguientes programas concurrentes. Asuma que "funcion" existe, y que los procesos son iniciados desde el programa principal.

P1	chan canal (double)
	process grano1 {
	int veces, i; double sum;
	for [veces= 1 to 10] {
	for [i = 1 to 10000]
	sum=sum+funcion(i);
		process grano2 {
		int veces; double sum;
		for [veces= 1 to 10] {
		receive canal (sum);
		printf (sum);
		}
		}
				P2	chan canal (double)
					process grano1 {
					int veces, i; double sum;
					for [veces= 1 to 10000] {
					for [i = 1 to 10]
					sum=sum+i;
					send canal (sum);
						process grano2 {
						int veces; double sum;
						for [veces= 1 to 10000] {
						receive canal (sum);
						printf (sum);
						}
						}

	send canal (sum); } }				} }	

a) Analice desde el punto de vista del número de mensajes.

b) Analice desde el punto de vista de la granularidad de los procesos.

c) Cuál de los programas le parece más adecuado para ejecutar sobre una arquitectura de tipo cluster de PCs? Justifique.

4. Indiqué por qué puede considerarse que existe una dualidad entre los mecanismos de monitores y pasaje de mensajes. Ejemplifique.

Existe dualidad entre monitores y PM porque cada uno de ellos puede simular al otro. Por ejemplo los simularemos usando procesos servidores y MP, como procesos activos en lugar de como conjuntos pasivos de procedures.

5. Dado el siguiente programa concurrente con memoria compartida:

```
x = 3; y = 2; z = 5;
co
  x = y * z //  z = z * 2   // y = y + 2x
oc
```

a) Cuáles de las asignaciones dentro de la sentencia co cumplen la propiedad de "A lo sumo una vez". Justifique claramente.

Siendo:
A: x = y * z B: z = z * 2 C: y = y + 2x

A tiene 2 referencias críticas (a y, a z), por lo tanto no cumple ASV. (además x es leída en C).
B no tiene referencia crítica y es leída por otro (en A se lee z), por lo tanto cumple ASV.
C tiene 1 referencia crítica (a x) y además es leída por otro proceso (en A se lee y), por lo tanto no cumple ASV.

b) Indique los resultados posibles de la ejecución. Justifique.
Nota 1: las instrucciones NO SON atómicas.
Nota 2: no es necesario que liste TODOS los resultados, pero sí todos los casos que resulten significativos.

Si se ejecutan en el orden A, B y C o A, C y B: x = 10; z = 10; y = 22

Si se ejecutan en el orden C, B y A o B, C y A: x = 80; z = 10; y = 8

Si se ejecutan C, A y B en dicho orden: x = 40; z = 10; y = 8

Si se ejecutan B, A y C en dicho orden: x = 20; z = 10; y = 42

Si se empieza a ejecutar A leyendo a y = 2, y en ese momento se ejecuta C leyendo a x = 3 (porque no terminó la asignación de A), y luego termina lo que falta de A y se ejecuta B: x = 10; z = 10; y = 8

Si se empieza a ejecutar A leyendo a y = 2, y en ese momento se ejecuta C leyendo a x = 3 (porque no terminó la asignación de A), y luego se ejecuta B y lo que falta de A: x = 20; z = 10; y = 8

6. a) Defina el concepto de "sincronización barrier". Cuál es su utilidad?

El punto de demora al final de cada iteración, en un programa concurrente, es una barrera a la que deben llegar todos antes de permitirles pasar, a esto se lo conoce como sincronización barrier. Es útil para poner un punto de encuentro entre los procesos de un programa concurrente a la espera de una determinada condición que le permita proseguir su curso normal. Dicha condición puede ser para mantener la integridad de los datos o bien para realizar una nueva iteración de los procesos.

b) Qué es una barrera simétrica?

Una barrera simétrica para n procesos se construye a partir de pares de barreras simples para dos procesos. Consiste en que cada proceso tiene un flag que setea cuando arriba a la barrera. Luego espera a que el otro proceso setee su flag y finalmente limpia la bandera del otro.

c) Describa "combining tree barrier" y "butterfly barrier". Marque ventajas y desventajas en cada caso.

Combining tree barrier: Se utiliza cuando se tiene un problema en el que se requiere un proceso (y procesador) extra y el tiempo de ejecución del coordinador es proporcional a n. La solución que brinda esta técnica se basa en combinar las acciones de workers y coordinador, haciendo que cada worker sea también coordinador. Por ejemplo, workers en forma de árbol: las señales de arribo van hacia arriba en el árbol, y las de continuar hacia abajo. Este tipo de solución es más eficiente para n grande).

Butterfly barrier: Surge debido a la forma del patrón de interconexión al combinar las barreras para dos procesos, para construir una barrera n procesos. Una butterfly barrier tiene $\log_2 n$ etapas. Cada Worker sincroniza con un Worker distinto en cada etapa. En particular, en la etapa s un Worker sincroniza con un Worker a distancia 2^{s-1}. Cuando cada Worker pasó a través de $\log_2 n$ etapas, todos los Workers deben haber arribado a la barrera y por lo tanto todos pueden seguir. Esta técnica sirve si n es potencia de 2, cuando n no es potencia de 2, es más eficiente usar dissemination barrier.

7. a) Describa brevemente en qué consisten los mecanismos de RPC y Rendezvous. Para qué tipo de problemas son más adecuados?

Los mecanismos de Remote Procedure Call [RPC] y Rendezvous son ideales para interacciones cliente/servidor. Ambas combinan aspectos de monitores y pasaje de mensaje sincrónico (SMP). Como con monitores, un módulo o proceso exporta operaciones, y las operaciones son invocadas por una sentencia call. Como con las sentencias de salida en SMP, la ejecución de call demora al llamador. La novedad de RPC y Rendezvous es que una operación es un canal de comunicación bidireccional desde el llamador al proceso que sirve el llamado y nuevamente hacia el llamador. En particular, el llamador se demora hasta que la operación llamada haya sido ejecutada y se devuelven los resultados. Es por esto que son más adecuados para problemas de tipo cliente/servidor ya que se precisa de una comunicación bidireccional.

b) Por qué es necesario proveer sincronización dentro de los módulos en RPC? Cómo puede realizarse esta sincronización?

Por sí mismo, RPC es puramente un mecanismo de comunicación. Aunque un proceso llamador y su server sincronizan, el único rol del server es actuar en nombre del llamador. Conceptualmente, es como si el proceso llamador mismo

estuviera ejecutando el llamado, y así la sincronización entre el llamador y el server es implícita. Pero esto no sucede así con los procesos en un módulo, es por eso que la sincronización debemos asegurarla mediante exclusión mutua y sincronización por condición, con el fin de garantizar la correcta ejecución tanto a los procesos server que están ejecutando llamados remotos como a otros procesos declarados en el módulo.
Esta sincronización puede realizarse dependiendo de dos casos:

I. Si los procesos en un módulo se ejecutan con exclusión mutua, es decir un solo un proceso activo a la vez, entonces las variables compartidas son protegidas automáticamente contra acceso concurrente; pero los procesos necesitan alguna manera de programar sincronización por condición. Para esto podríamos usar automatic signalig (await B) o variables condición.
II. Si los procesos en un módulo pueden ejecutar concurrentemente (al menos conceptualmente), necesitamos mecanismos para programar tanto exclusión mutua como sincronización por condición. En este caso, cada módulo es en sí mismo un programa concurrente, de modo que podríamos usar cualquiera de los métodos (semáforos, monitores, o incluso Rendezvous).

c) Qué elementos de la forma general de rendezvous no se encuentran en el lenguaje ADA?

ADA no provee la expresión de Scheduling "e_i" la cual se usa para alterar el orden de servicio de invocaciones por default en las guardas.

8. a) Cuál es el objetivo de la programación paralela?

El objetivo de la programación paralela el de resolver un problema en el menor tiempo (o un problema más grande en aproximadamente el mismo tiempo) usando una arquitectura multiprocesador en la que se pueda distribuir la tarea global en tareas que puedan ejecutarse en distintos procesadores.

b) Defina las métricas de speedup y eficiencia. Cuál es el significado de cada una de ellas (qué miden)? Ejemplifique.

La métrica de speedup representa la ganancia que tenemos al usar más procesadores para un mismo problema. Se define por la siguiente fórmula:

$S = T_s / T_p$

I. p es el número de procesadores.
II. T_s es el tiempo de ejecución del algoritmo secuencial.
III. T_p es el tiempo de ejecución del algoritmo paralelo con p procesadores.

El rango de valores en general se encuentra entre 0 y p. Aunque hay casos de speedup superlineal, es decir mayor que p.
Por ejemplo si obtenemos un speedup S = p estamos frente a un speedup lineal, esto quiere decir que duplicando el número de procesadores duplicamos la velocidad, lo cual es ideal.

La métrica de eficiencia mide la fracción de tiempo en que los procesadores son útiles para el cómputo. Se define por la siguiente fórmula:

$E = S / p = T_s / pT_p$

El rango de valores se encuentra entre 0 y 1, dependiendo de la efectividad de uso de los procesadores.
Por ejemplo: Algoritmos con speedup linear y algoritmos que corren en un procesador simple, tienen eficiencia 1. Muchos algoritmos difíciles de paralelizar tienen eficiencia cercana a 1/log p, que se aproxima a cero a medida que el número de procesadores se incrementa.

c) Suponga que la solución a un problema es paralelizada sobre *p* procesadores de dos maneras diferentes. En un caso, la eficiencia está regido por la función E=1/p y en el otro por la función E= $1/p^2$. Cuál de las dos soluciones se comportará más eficientemente al crecer la cantidad de procesadores? Justifique.

Claramente se observa que a partir de p=2, E1=1/p será mayor que E2= $1/p^2$, observemos los siguientes casos:

I. p = 1, tenemos E1 = 1/1 = 1 y E2 = 1/1 = 1
II. p = 2, tenemos E1 = 1/2 = 0,5 y E2 = 1/4 = 0,25
III. p = 3, tenemos E1 = 1/3 = 0,33 y E2 = 1/9 = 0,11

Como se puede apreciar, a partir de p=2 E1 se mantiene más eficiente que E2, pero sin embargo ambos van a decrecer conforme p crezca, por lo tanto ninguno de los dos se comportará más eficiente al crecer la cantidad de procesadores.

d) Suponga que el tiempo de ejecución de un algoritmo secuencial es de 10000 unidades de tiempo, de las cuales sólo el 80% corresponde a código paralelizable. Cuál es el límite en mejora que puede obtenerse paralelizando el algoritmo?

Si el tiempo de ejecución de un algoritmo secuencial es de 10000 unidades y el porcentaje de unidades paralelizables es del 80%, quiere decir que 2000 unidades (el 20%) deben ejecutarse secuencialmente, por lo tanto el tiempo mínimo esperable es de 2000 para un procesador. Por esta razón nos conviene utilizar una cantidad de procesadores tal que los mantenga a todos ocupados esa cantidad de tiempo, es decir en este caso conviene tener 5 procesadores porque si tuviéramos más un procesador estaría trabajando 2000 unidades de tiempo y los demás terminarían de ejecutar las instrucciones antes, y tendrían que esperarlo.

9. Explique sintéticamente los 7 paradigmas de interacción entre procesos en programación distribuida. En cada caso ejemplifique, indique qué tipo de comunicación por mensajes es más conveniente y cuál es la arquitectura de hardware que se ajusta mejor? Justifique sus respuestas.

I. Servidores Replicados: Los servidores manejan (mediante múltiples instancias) recursos compartidos tales como dispositivos o archivos. Ejemplo: File servers con múltiples clientes y una instancia de servidor por cliente (o por File, o por Unidad de almacenamiento).
II. Algoritmos Heartbeat: Los procesos periódicamente deben intercambiar información con mecanismos tipo send/receive. Ejemplo: Modelos biológicos / Problema de los N Cuerpos / Modelos de simulación paramétrica.
III. Algoritmos Pipeline: La información recorre una serie de procesos utilizando alguna forma de receive/send. Se supone una arquitectura de procesos/procesadores donde la salida de uno es entrada del siguiente. Los procesadores pueden distribuirse DATOS o FUNCIONES. Ejemplo: Redes de Filtros, Tratamiento de Imágenes.
IV. Probes (send) y Echoes (receive): La interacción entre los procesos permite recorrer grafos o árboles (o estructuras dinámicas) diseminando y juntando información. Un ejemplo clásico es recuperar la topología activa de una red móvil o hacer un broadcast desde un nodo cuando no se “alcanzan” o “ven” directamente todos los destinatarios.
V. Algoritmos Broadcast: Permiten alcanzar una información global en una arquitectura distribuida. Sirven para toma de decisiones descentralizadas. En general en sistemas distribuidos con múltiples procesadores, las comunicaciones colectivas representan un costo crítico en tiempo. Un ejemplo típico es la sincronización de relojes en un Sistema Distribuido de Tiempo Real.
VI. Token Passing: En muchos casos la arquitectura distribuida recibe una información global a través del viaje de tokens de control o datos. La arquitectura puede ser un anillo, caso en el cual el manejo se asemeja a un pipeline, pero también puede ser cualquier topología (tipo objetos distribuidos). Los tokens normalmente habilitan el control para la toma de decisiones distribuidas.
VII. Manager/Workers: Implementación distribuida del modelo de bag of tasks. Un procesador controla los datos y/o procesos a ejecutarse y múltiples procesadores acceden a él para acceder a datos/procesos y ejecutar las tareas distribuidas.

10. Suponga n^2 procesos organizados en forma de grilla cuadrada. Cada proceso puede comunicarse solo con los vecinos izquierdo, derecho, de arriba y de abajo (los procesos de las esquinas tienen solo 2 vecinos, y los otros en los bordes de la grilla tienen 3 vecinos). Cada proceso tiene inicialmente un valor local v.

a) Escriba un algoritmo heartbeat que calcule el máximo y el mínimo de los n^2 valores. Al terminar el programa, cada proceso debe conocer ambos valores.

b) Analice la solución desde el punto de vista del número de mensajes.

c) Puede realizar alguna mejora para reducir el número de mensajes?