PROGRAMACION CONCURRENTE 2010 (Planes 2003-2007) – Final – 16/3/2011

En todos los casos, responda con claridad y sintéticamente. En los casos que corresponda, NO SE CONSIDERARAN RESPUESTAN SIN JUSTIFICAR. Tiempo Máximo 2 hs 15 min.

Parte A : Definiciones y preguntas conceptuales que debieran ser contestadas brevemente (15 puntos)

1- 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.

2- Defina sincronización entre procesos y mecanismos de sincronización.

Sincronización entre procesos: Es la posesión de información acerca de otro proceso para coordinar actividades. El objetivo de la sincronización es restringir las historias de un programa concurrente sólo a las permitidas.

Mecanismos de sincronización:

I. **Exclusión mutua:** Consiste en asegurar que sólo un proceso tenga acceso a un recurso compartido en un instante de tiempo, evitando que dos o más procesos puedan encontrarse en la misma sección crítica al mismo tiempo.
II. **Sincronización por condición:** Asegura que la ejecución de un proceso se bloquee si es necesario, hasta que sea verdadera una condición dada.

3- Defina el concepto de no determinismo. Ejemplifique.

El no determinismo en los programas concurrentes implica que, al ejecutarse las instrucciones que involucran a los mismos, puedan darse distintos resultados de salida sobre los mismos datos de entrada.

Ejemplo:

```
process imprime10 {
   for [i=1 to n]
   write(i);
}
```

```
process imprime1 [i=1 to 10] {
   write(i);
}
```

Al ejecutarse concurrentemente una vez el proceso imprime10 y los 10 procesos imprime1 los resultados en pantalla se intercalarán. Si se vuelve a ejecutar una segunda vez, puede darse que los resultados se intercalen de otra manera distinta, por lo tanto estaríamos frente a una situación de no determinismo.

4- En qué consiste la propiedad de "A lo sumo una vez" y qué efecto tiene sobre las sentencias de un prog. concurrente?

Una sentencia de asignación x = e satisface la propiedad de "A lo sumo una vez" si:

I. e contiene a lo sumo una referencia crítica (referencia a una variable que es modificada por otro proceso) y x no es referenciada por otro proceso, o
II. e no contiene referencias críticas, en cuyo caso x puede ser leída por otro proceso

En una expresión que no se encuentra en sentencias de asignación, se dice que cumple la propiedad de a lo sumo una vez si dicha expresión no contiene más de una referencia crítica.

El efecto que tiene sobre las sentencias de un programa concurrente es que si una sentencia de asignación cumple la propiedad ASV, entonces su ejecución parece atómica, pues la variable compartida será leída o escrita sólo una vez.

5- Por qué las propiedades de vida dependen de la política de scheduling? Cuándo una política de scheduling es fuertemente fair?

Las propiedades de vida en los programas concurrentes hacen referencia a pedidos de servicio que eventualmente serán atendidos, que un mensaje eventualmente alcanzará su destino, y que un proceso eventualmente entrará a su sección crítica. Es por eso que se ven afectadas por las políticas de scheduling, ya que estas son las que determinan cuáles acciones atómicas elegibles son las próximas en ejecutarse.

Una política de scheduling es fuertemente fair si:

I. es incondicionalmente fair (es decir que toda acción atómica incondicional que es elegible eventualmente es ejecutada) y
II. toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada si su guarda es true con infinita frecuencia.

6- Describa la técnica de passing the baton.

Es una técnica que proporciona una solución a problemas como el de los lectores y escritores usando sincronización por condición. Esta técnica de programación se llama así por la manera en que los semáforos son señalizados. Cuando un proceso está ejecutando su sección crítica, puede pensarse que posee la "posta" o "baton", esto significa que tiene permiso de ejecución. Cuando este proceso llega al fragmento SIGNAL, pasa la posta a otro. Si alguno está esperando por una condición que es ahora verdadera, la posta se le otorga al mismo. Este a su vez, ejecuta su SC y la pasa a otro. Cuando no hay procesos esperando por la condición verdadera, la posta se pasa al próximo proceso que trate de entrar en su SC por primera vez.

7- En qué consiste la sincronización barrier? Mencione alguna de las soluciones posibles usando variables compartidas.

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.

Soluciones posibles usando variables compartidas:

I. Contador Compartido
II. Flags y Coordinadores
III. Arboles
IV. Barreras Simétricas
V. Butterfly Barrier

8- Qué se entiende x arquitectura de grano grueso? Es más adecuada para programas con mucha o poca comunicación?

Una arquitectura con pocos procesadores muy poderosos, es una arquitectura de grano grueso. Es por eso que este tipo de arquitectura se adecúa más a programas con poca comunicación y con mucho cómputo.

9- Qué significa que un problema sea de "exclusión mutua selectiva" (EMS)? El problema de los filósofos es de EMS? Por qué? Si en lugar de 5 filósofos fueran 3, el problema seguiría siendo de EMS? Por qué?

En los problemas de exclusión mutua selectiva, cada proceso compite por sus recursos no con todos los demás procesos sino con un subconjunto de ellos. Dos casos típicos de dicha competencia se producen cuando los procesos compiten por los recursos según su tipo de proceso o por su proximidad.

El problema de los filósofos es de EMS, ya que se basa en que 5 filósofos tratan de comer usando dos tenedores, el de su izquierda y el de su derecha; cuando sólo hay 5 tenedores en total, por lo que se tienen procesos (un filósofo) que compiten con un subconjunto de procesos (los filósofos que lo rodean) por recursos (el tenedor a la izquierda y a la derecha de cada filósofo).

Si en lugar de 5 filósofos fueran 3 el problema dejaría de ser de EMS ya que cada filósofo competiría con todos los demás filósofos por los mismos recursos.

10- En qué consiste la comunicación guardada (introducida por CSP) y cuál es su utilidad? Describa cómo es la ejecución de sentencias de alternativa e iteración que contienen comunicaciones guardadas.

El lenguaje formal CSP (Communicating Sequential Processes) de Hoare, fue uno de los desarrollos fundamentales en programación concurrente. La idea básica de Hoare fue la comunicación guardada, es decir pasaje de mensajes con waiting selectivo. Consiste en la idea de tener canales simples como links directos entre dos procesos (en vez de mailbox global) y sentencias de entrada, salida bloqueantes (?, !) como único medio por el cual se comunican los procesos. Es útil para problemas como el de copiar con un buffer limitado, el de asignación de recursos y el de intercambio de valores.

Sentencias que contienen comunicaciones guardadas (if – de alternativa | do – de iteración):

```
if B1; comunicación1->S1;
   B2; comunicación2->S2;
fi
```

```
do B1; comunicación1->S1;
   B2; comunicación2->S2;
od
```

a) Primero, se evalúan las expresiones booleanas:
   I. Si todas las guardas fallan, el if termina sin efecto.
   II. Si al menos una guarda tiene éxito, se elige una de ellas (no determinísticamente).
   III. Si algunas guardas se bloquean, se espera hasta que alguna de ellas tenga éxito.
b) Segundo, luego de elegir una guarda exitosa, se ejecuta la sentencia de comunicación en la guarda elegida.
c) Tercero, se ejecuta la sentencia Si.

La ejecución del do es similar sólo que se repite hasta que todas las guardas fallen.

Parte B: Interpretación de código (10 puntos: 3 + 3 + 4)

11- Dado el siguiente programa concurrente con variables compartidas:

```
x = 4; y = 2; z = 3;
co
  x = y * z //  z = z * 2    // y = y + 2x
oc
```

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

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 = 6; z = 6; y = 14

Si se ejecutan en el orden C, B y A o B, C y A: x = 60; z = 6; y = 10

Si se ejecutan C, A y B en dicho orden: x = 30; z = 6; y = 10

Si se ejecutan B, A y C en dicho orden: x = 12; z = 6; y = 26

Si se empieza a ejecutar A leyendo a y = 2, y en ese momento se ejecuta C leyendo a x = 4 (porque no terminó la asignación de A), y luego termina lo que falta de A y se ejecuta B: x = 6; z = 6; y = 10

Si se empieza a ejecutar A leyendo a y = 2, y en ese momento se ejecuta C leyendo a x = 4 (porque no terminó la asignación de A), y luego se ejecuta B y lo que falta de A: x = 12; z = 6; y = 10

12- Suponga los siguientes programas concurrentes. Asuma que EOS es un valor especial que indica el fin de la secuencia de mensajes, y que los procesos son iniciados desde el programa principal.

P1

```
chan canal (double)
process Genera {
   int fila, col;   double sum;
   for [fila= 1 to 10000]
      for [col = 1 to 10000]
         send canal (a(fila,col));
   send canal (EOS)   }
```

```
process Acumula {
   double valor, sumT;
   sumT=0;
   receive canal (valor);
   while valor<>EOS {
      sumT = sumT + valor
      receive canal (valor); }
   printf (sumT);
}
```

P2

```
chan canal (double)
process Genera {
   int fila, col;   double sum;
   for [fila= 1 to 10000] {
      sum=0;
      for [col = 1 to 10000]

      sum=sum+a(fila,col);
      send canal (sum);
}
   send canal (EOS)
}
```

```
process Acumula {
   double valor, sumT;
   sumT=0;
   receive canal (valor);
   while valor<>EOS {
      sumT = sumT + valor
      receive canal (valor);
   }
   printf (sumT);
}
```

a) Qué hacen los programas?

Ambos programas realizan la suma total de todos los elementos de una matriz y la imprimen en pantalla.

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

P1 realiza 10000^2 mensajes porque el proceso Genera manda el valor de cada elemento de la matriz, y P2 realiza 10000 mensajes porque el proceso Genera manda el valor de las sumas totales de cada columna de la matriz.

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

Desde el punto de vista de la cantidad de procesos, diría que son procesos de grano grueso, ya que en los dos programas son pocos los procesos que interactúan para resolver el problema y cada uno realiza mucho trabajo.

Desde el punto de vista en cuanto a la relación de comunicación y cómputo de los procesos podría decirse que los procesos de P1 son de grano fino debido a que realizan muchos mensajes y el cómputo sólo lo realiza el proceso acumula, en cambio P2 realiza muchos menos mensajes que P1 (aunque siguen siendo bastantes con respecto a otras soluciones posibles), y el cómputo es distribuido entre los dos procesos; por esto comparado con P1 diría que P2 es de grano medio o más grueso que P1.

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

Como P1 realiza todo su cómputo en un solo proceso, y P2 divide el cómputo entre los dos procesos, el más adecuado sería P2; aunque de estas dos soluciones en realidad ninguna aprovecharía una arquitectura de este tipo, ya que sería conveniente hacer uso de más procesos independientes para realizar el cómputo dividido de una manera más eficiente.

13- Dados los siguientes dos segmentos de código, indicar para cada uno de los ítems si son equivalentes o no. Justificar cada caso (de ser necesario dar ejemplos).

Segmento 1	Segmento 2

...
int cant=1000;
DO (cant < -10); datos?(cant) →
Sentencias1
□ (cant > 10); datos?(cant) →
Sentencias2
□ (INCOGNITA); datos?(cant) →
Sentencias3
END DO
...
	...
	int cant=1000;
	While (true)
	{ IF (cant < -10); datos?(cant) →
	Sentencias1
	□ (cant > 10); datos?(cant) →
	Sentencias2
	□ (INCOGNITA); datos?(cant) →
	Sentencias3
	END IF
	}
	...

Viendo que en el segmento 1 se utiliza un do y en el segmento 2 se utiliza un if dentro de un loop infinito, para que dichos segmentos trabajen de forma equivalente se debe lograr que el do también sea infinito. Para esto debe contemplar todos los posibles valores de cant en sus guardas, ya que si no es así, algún valor de cant no contemplado en ninguna guarda hará obligar a que la sentencia do termine.

a) *INCOGNITA* equivale a: *(cant = 0),*

No son equivalentes porque si en algún momento 0>cant≥-10 o 10≥cant>0, todas las guardas serán falsas, lo que hará que el do termine su ejecución y no sea infinito.

b) *INCOGNITA* equivale a: *(cant > -100)*

Los segmentos son equivalentes porque aquí el do contempla todos los valores de cant asegurándose, lo que asegura que el do sea infinito.

c) *INCOGNITA* equivale a: *((cant > 0) or (cant < 0))*

No son equivalentes porque si en algún momento cant=0, todas las guardas serán falsas, lo que hará que el do termine su ejecución y no sea infinito.

d) *INCOGNITA* equivale a: *((cant > -10) or (cant < 10))*

Los segmentos son equivalentes porque aquí el do contempla todos los valores de cant asegurándose, lo que asegura que el do sea infinito.

e) *INCOGNITA* equivale a: *((cant >= -10) or (cant <= 10))*

Los segmentos son equivalentes porque aquí el do contempla todos los valores de cant asegurándose, lo que asegura que el do sea infinito.

Parte C: Temas para desarrollar (12 puntos: 3 + 5 + 4)

14- 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¡" la cual se usa para alterar el orden de servicio de invocaciones por default en las guardas.

15- 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. 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.

16- 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) Mencione al menos 4 problemas en los cuales Ud. entiende que es conveniente el uso de técnicas de programación paralela.

I. Cálculo científico. Modelos de sistemas (meteorología, movimiento planetario, etc.)
II. Gráficos, procesamiento de imágenes, efectos especiales, procesamiento de video, realidad virtual
III. Problemas combinatorios y de optimización lineal o no lineal. Modelos econométricos

c) Defina las métricas de speedup y eficiencia. Cuál es el significado de cada una de ellas (qué miden)? Cuál es el rango de valores posibles de cada uno? 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.

d) Suponga que la solución a un problema es paralelizada sobre *p* procesadores de dos maneras diferentes. En un caso, el speedup (S) está regido por la función S=p/3 y en el otro por la función S=p-3. Cuál de las dos soluciones se comportará más eficientemente al crecer la cantidad de procesadores? Justifique claramente.

Claramente se observa que a partir de p=5, el speedup S2=p-3 será mayor que S1=p/3, por lo tanto lograremos una mayor eficiencia de la solución al problema a medida que p >= 5, observemos los siguientes casos:

I. p = 4, tenemos para S1 S1 = 4/3 = 1,33 con E1 = 1,33/4 = 0,33 y para S2 S2 = 4-3 = 1 con E2 = 1/4 = 0,25

II. p = 5, tenemos para S1 S1 = 5/3 = 1,66 con E1 = 1,66/4 = 0,41 y para S2 S2 = 5-3 = 2 con E2 = 2/4 = 0,5

III. p = 6, tenemos para S1 S1 = 6/3 = 2 con E1 = 2/4 = 0,5 y para S2 S2 = 6-3 = 3 con E2 = 3/4 = 0,75

Como se puede apreciar, con p=4, S1 es más eficiente; pero con p=5 y p=6, es decir p >=5, S2 es más eficiente.

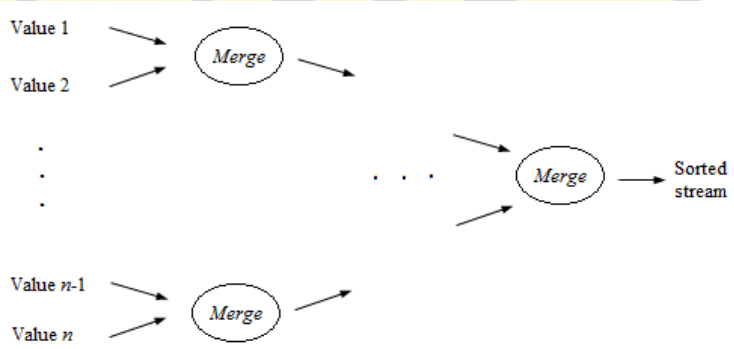
e) Suponga que el tiempo de ejecución de un algoritmo secuencial es de 10000 unidades de tiempo, de las cuales sólo el 80% corresponde a código paralelizable. Cuál es el límite en mejora que puede obtenerse paralelizando el algoritmo? Justifique

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.

Parte D: Ejercicio a resolver (8 puntos)

17- Suponga los siguientes métodos de ordenación de menor a mayor para n valores (n par y potencia de 2), utilizando pasaje de mensajes:

i- Un pipeline de filtros. El primero hace input de los valores de a uno por vez, mantiene el mínimo y le pasa los otros al siguiente. Cada filtro hace lo mismo: recibe un stream de valores desde el predecesor, mantiene el más chico y pasa los otros al sucesor.



ii- Una red de procesos filtro (como la de la figura).

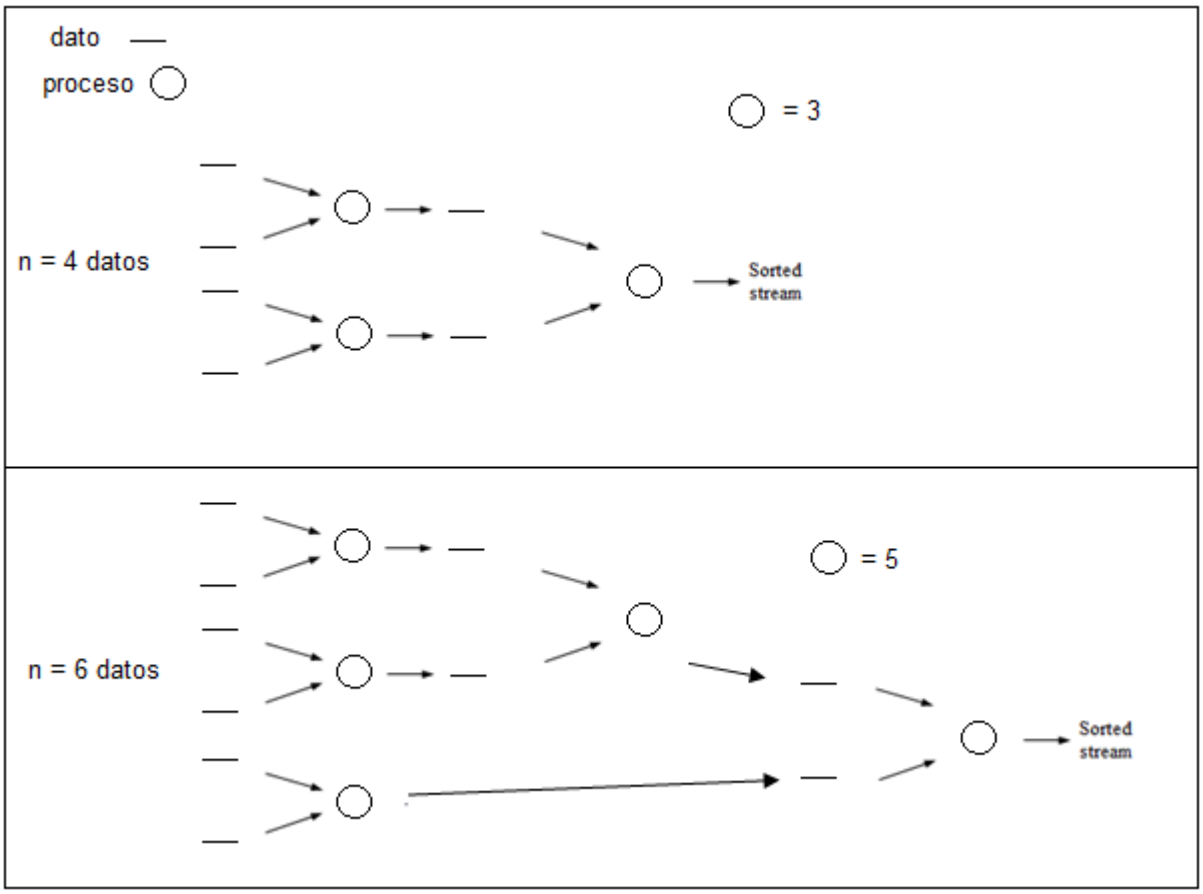
iii- Odd/even Exchange sort. Hay n procesos P[1:n], Cada uno ejecuta una serie de rondas. En las rondas "impares", los procesos con número impar P[impar] intercambian valores con P[impar+1]. En las rondas "pares", los procesos con número par P[par] intercambian valores con P[par+1] (P[1] y P[n] no hacen nada en las rondas "pares"). En cada caso, si los números están desordenados actualizan su valor con el recibido.

Nota: Cada proceso tiene almacenamiento local sólo para 2 valores (el próximo y el mantenido hasta ese momento).

a) Cuántos procesos son necesarios en i e ii? Justifique.

Algoritmo i: Se necesitan n procesos ya que cada proceso almacena el valor más pequeño de todos los que recibe, es decir que como se envían n valores, se necesitan n procesos.

Algoritmo ii: Como en un principio se necesitan procesar n datos, y cada proceso recibe dos, entonces se necesitan n/2 procesos. Por consiguiente necesitaremos además n/2 - 1 procesos para hacer un sort del resultado de los anteriores. Por lo tanto tenemos n/2 + n/2 -1 = n-1, y entonces se necesitan n-1 procesos en total. Por ejemplo observemos dos árboles uno con n=4 y otro con n=6, la cantidad de nodos que representan procesos es 3 y 5 respectivamente.



b) Cuántos mensajes envía cada algoritmo para ordenar los valores? Justifique.

Algoritmo i: Asignándole a cada proceso un número entero consecutivo, siendo el primer proceso el 1, el segundo el 2, así sucesivamente hasta el último proceso n, tenemos que:
Al proceso 1 le llegan n mensajes, y envía n-1 mensajes, porque se queda con el valor más chico de todos los que le llegan y va enviando los demás.
Al proceso 2 le llegan n-1 mensajes y envía n-2 mensajes, por la misma razón del proceso anterior.
Y así sucesivamente, hasta el proceso n, el cual recibe 1 (n-(n-1)) mensaje y no manda ningún (n-n) mensajes.
Observando dicho patrón en la sucesión de envíos, podemos afirmar que el número total de mensajes que se envían es:

$$\sum_{i=1}^{n} i = \frac{n\ (n+1)}{2}$$

Algoritmo ii: Como por cada proceso se realiza un solo envío y al principio se envían n mensajes (1 por cada elemento), entonces la cantidad de mensajes que se envían es (n-1) + n = 2n - 1.

Algoritmo iii: Cada proceso envía uno y recibe uno dependiendo de que si la ronda es par o impar. Además para calcular u ordenar, en este algoritmo se necesitan n rondas en el peor de los casos. Por cada ronda se mandan n mensajes y se reciben n mensajes para que cada proceso (par o impar dependiendo de la ronda) intercambie su valor con otro proceso. Es por todo esto que en este algoritmo se mandan n^2.

c) En cada caso, cuáles mensajes pueden ser enviados en paralelo (asumiendo que existe el hardware apropiado) y cuáles son enviados secuencialmente? Justifique.

Algoritmo i: Cada proceso puede enviar su mensaje de salida en paralelo a otro proceso, siempre y cuando haya recibido el input enviado por su predecesor.

Algoritmo ii: Cualquier proceso puede enviar su mensaje de salida en paralelo a otro proceso, siempre y cuando haya recibido los dos inputs provenientes de sus dos nodos hijos.

Algoritmo iii: Por ronda se mandarán n/2 mensajes porque solo los de ronda par o ronda impar enviarán un mensajes con el dato más grande al mismo tiempo (también podría suceder lo mismo y recibir n/2 simultáneamente). Es por esto que este algoritmo puede enviar $n^2/2$ mensajes en paralelo.

d) Cuál es el tiempo total de ejecución de cada algoritmo? Asuma que cada operación de comparación o de envío de mensaje toma una unidad de tiempo. Justifique.

Algoritmo i: Utilizando el cálculo de la cantidad de mensajes del punto a) y razonando que cada vez que se envía un mensaje se realiza una comparación antes, tenemos que el tiempo de ejecución será igual a la cantidad de mensajes multiplicada por 2 unidades de tiempo (1 comparación + 1 mensaje):

$2 \sum_{i=1}^{n} i = 2 \quad n \quad (n+1) = n \quad (n+1) \Rightarrow O(n^2)$

2

Algoritmo ii: Por la misma razón que el algoritmo anterior tenemos:

$2\,(2n-1) = 4n-1 \Rightarrow O(n)$

Algoritmo iii: Cada proceso en cada ronda hace una comparación, envía, recibe y hace una asignación, por eso tenemos que por cada proceso se tarda 4 unidades. Como se necesitan n rondas en el peor de los casos, se realizará 4n unidades de tiempo en total, por lo tanto es de O(n).

TOTAL = 45 puntos. Para aprobar se requieren 27 puntos, correspondientes al menos a 3 de las partes, una de las cuales debe ser la parte D