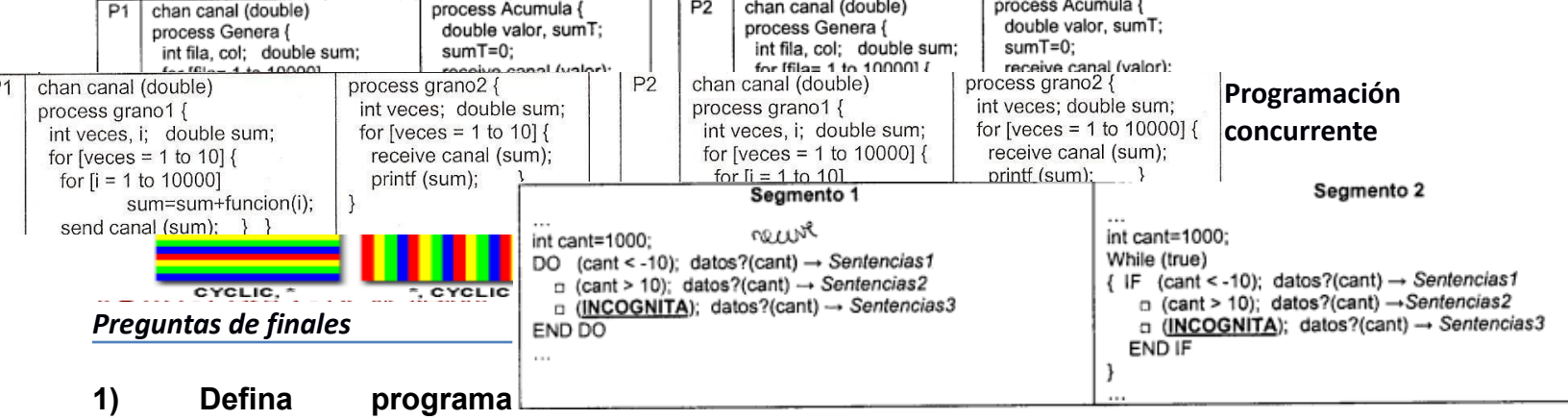


Preguntas de finales

1) Defina programa concurrente, programa paralelo y programa distribuido?

Un programa concurrente consiste en un conjunto de tareas o procesos secuenciales que pueden ejecutarse intercalándose en el tiempo y que cooperan para resolver un problema. Básicamente, es un concepto de software que dependiendo de la arquitectura subyacente da lugar a las definiciones de programación paralela o distribuida.

La programación paralela consiste en un programa concurrente que se ejecuta sobre múltiples procesadores que pueden tener una memoria compartida y que son utilizados para incrementar la performance de un programa concurrente.

La programación distribuida es un caso especial de la programación concurrente en la que se cuenta con varios procesadores pero no se posee una memoria compartida y la comunicación está dada por el pasaje de mensajes.

2)

a) ¿A qué se denomina propiedad del programa? ¿Qué son las propiedades de seguridad y vida? Ejemplificar.

Una *propiedad* es un atributo de un programa que es verdadero en todas sus posibles historias. Como un programa concurrente está formado por procesos una historia es cada interleaving de acciones atómicas que puede darse en una ejecución del programa.

Una propiedad de *seguridad* es aquella que asegura que nada malo sucede durante la ejecución del programa. Son ejemplos de propiedades de seguridad la ausencia de deadlock, lo malo es tener procesos esperando condiciones que no ocurrirán nunca; la exclusion mutua, donde lo malo es tener más de un proceso ejecutando sus secciones criticas al mismo tiempo.

Una propiedad de *vida* es aquella que asegura que algo bueno eventualmente ocurre durante la ejecución. Son ejemplos de propiedad de vida que un proceso eventualmente alcance su SC o que un mensaje llegue a destino pero estas propiedades dependen de la política de scheduling.

a) **Defina fairness. Relacionar dicho concepto con las políticas de scheduling.**

Fairness trata de asegurar que un proceso tenga la posibilidad de avanzar sin importar lo que hagan el resto de los procesos. Dado que el avance de un proceso está dado por la elección de la próxima sentencia atómica que él debe ejecutar, el concepto de fairness está relacionado estrechamente con las *políticas de scheduling* ya que estas son las que deciden cual es la próxima sentencia atómica a ejecutarse.

b) Describa los distintos tipos de fairness

Fairness incondicional: Toda acción atómica incondicional que es elegible es eventualmente ejecutada.

Fairness débil: Es incondicionalmente fair y toda acción atómica condicional que se vuelve elegible es ejecutada si su guarda se pone en true y se mantiene así hasta que es vista por el proceso que ejecuta la acción atómica condicional.

Fairness fuerte: Es incondicionalmente fair y toda acción atómica condicional que se vuelve elegible es ejecutada si su guarda es true con infinita frecuencia.

c) **¿Cuáles son las propiedades que debe cumplir un protocolo de E/S a una sección crítica?**

Estos protocolos deben cumplir con 4 propiedades:
Exclusión mutua: Solo un proceso a la vez puede estar ejecutando su SC. Es una propiedad de seguridad.

Ausencia de deadlock: Si uno o más procesos están intentando entra a su SC al menos uno de ellos tendrá éxito. Es una propiedad de seguridad.

Ausencia de demora innecesaria: si un proceso está intentando acceder a su SC y ningún otro proceso está ejecutando su SC o está en su SNC o terminaron, el primero no está impedido de ingresar. Es una propiedad de seguridad.

Eventual entrada: Un proceso que está intentando entrar a su SC eventualmente lo hará. Es una propiedad de vida.

d) **Cuáles son los defectos que presenta la sincronización por busy waiting? Diferencie esta situación respecto de los semáforos.**

La sincronización por busy waiting es ineficiente cuando los procesos son implementados por multiprogramación ya que se mantendrá ocupado un procesador realizando un spinning cuando este podría ser usado por otro proceso. También tiene el problema de que al estar chequeando por una variable compartida que es modificada por otros procesos muchas veces ese valor se encuentra invalidado en la cache lo que requiere que se acceda a memoria; este problema es muy notorio en implementaciones de acceso a SC.
Busy waiting y semáforos son herramientas que sirven para sincronización pero son distintas ya que semáforos evitan los problemas que tiene busy waiting. Al bloquearse en un semáforo un proceso no consume tiempo de procesamiento hasta que no tenga posibilidad de ejecutarse, en cuyo caso, es puesto en la cola de listos para poder usar el procesador.

e) **Explique la semántica de la instrucción de grano grueso AWAIT y su relación con las instrucciones Test & Set o Fetch & Add.**

<AWAIT B -> S;> Sintaxis de la sentencia await de grano grueso. Semánticamente, B es una expresión booleana que especifica una condición de demora y S es una secuencia de sentencias que se garantiza que terminan. También, se garantiza que B es true al momento de ejecutar S, y ningún estado interno de S es visible para los otros procesos. La relación entre la instrucción de grano grueso AWAIT y las instrucciones T&S y F&A es que estas últimas están presentes en casi todos los procesadores y son utilizadas para hacer atómico el AWAIT de grano grueso implementando los protocolos de E/S de la sección critica con los que se asegura la atomicidad del AWAIT.

3) Definir el problema general de alocación de recursos y su resolución mediante una política SJN. ¿Minimiza el tiempo promedio de espera? ¿Es fair? Si no lo es, plantee una alternativa que lo sea.

El problema general de alocación de recursos se da cuando varios procesos quieres acceder a unidades de un recurso compartido. Un proceso pide una o más unidades del recurso ejecutando un request con la cantidad de unidades que solicita y su identificación. Un request solo puede ser satisfecho si la cantidad de unidades disponibles es suficiente para satisfacer el request de lo contrario se demora hasta que haya suficientes unidades disponibles. Después de usar los recursos alocados, un proceso los retorna ejecutando la operación release.

Las operaciones de request y release deben ser atómicas ya que ambas acceden a la representación de los recursos. Se puede utilizar la técnica de passing the baton para el request y el release:

Request (parámetros): P (e)
if (request no puede satisfacerse) {DELAY}
Tomar las unidades;
SIGNAL;

Release (parámetros): P (e);
Retorna unidades
SIGNAL;

Su resolución mediante SJN se basa en varios procesos que requieren un único recurso compartido utilizando request como antes pero con los parámetros id y time siendo este el tiempo que retendrá el recurso para su uso. Por lo tanto, al ejecutar un request si el recurso está libre es inmediatamente alocado al proceso y en caso contrario, el proceso se demora. Cuando el recurso es liberado, utilizando el release, es alocado al proceso demorado (si lo hay) que tiene el mínimo valor de time. Si dos o más procesos demorados tienen el mismo valor de time se le asigna al que más tiempo ha estado demorado.
La política de SJN minimiza el tiempo promedio de ejecución pero es unfair ya que un proceso podría ser demorado para siempre si hay una corriente continua de request con time menor. Esta política puede modificarse para evitar esto utilizando la técnica de aging que consiste en darle preferencia a un proceso que ha estado demorado un largo tiempo. (Poner un valor fijo para determinar si un proceso espero mucho o implementar la variante del SJN, Highest Response Ratio Next que pone la prioridad como 1 + tiempo espera/tiempo ejecución).

1) **¿En qué consiste la técnica de passing the baton? Aplicar este concepto a la resolución del problema de lectores y escritores.**

Passing the baton es una técnica general para implementar sentencias await arbitrarias y para proveer un orden en la ejecución de los procesos. Consiste en que mientras un proceso se encuentre dentro de su SC mantiene el baton o derecho a ejecutar y cuando termina le pasa el baton al próximo proceso demorado si es que lo hay o al primer proceso que solicite el acceso a la SC. Le pasa el baton primero al proceso demorado para evitar que en caso de hacerle V y despertarlo, otro proceso le gane el acceso antes de que él pueda ejecutar. Es decir, se usa para despertar a los procesos demorados y que ejecuten en orden.
F1: <Si> y F2: <AWAIT Bj Sj> formas posibles de las acciones atómicas en una solución que utiliza passing the baton.
Ahora, sea e un semáforo inicializado en uno que se utiliza para controlar la entrada a las sentencias atómicas. Además de asociar un semáforo bj y un contador dj cada uno representando una guarda diferente Bj; estos son todos inicialmente 0. El semáforo bj se utiliza para demorar los procesos que esperan por la condición Bj y dj cuenta la cantidad de procesos demorados en bj. Con estos datos podemos ver cómo quedan formadas las sentencias atómicas con la técnica de passing the baton.

F1: P(e); S_i; SIGNAL	F2: P(e); if not (B_j){ d_j:=d_j+1; V(e); P(b_j)} S_j; SIGNAL	SIGNAL: if (B_1 and d_1>0) {d_1:= d_1 -1; V(b_1)} Elseif (B_2 and d_2>0) {d_2 := d_2 -1; V(b_2)} … Else V(e);

Para el caso de los lectores y escritores se usa passing de baton ya que con los semáforos no todas las sentencias await se pueden implementar. Particularmente en este caso, los protocolos de acceso

a la SC en ambos procesos comparten una condición pero los escritores agregan una más por lo que es imposible distinguirlas usando semáforos.
(Ver algoritmo aparte)

2) **Explique el concepto de broadcast y sus dificultades de implementación en un ambiente distribuido, con pasaje de mensajes sincrónico y asincrónico.**

El concepto de broadcast consiste en enviar concurrentemente un mismo mensaje a varios procesos.
En un ambiente compartido es más fácil que la información de un proceso pueda ser conocida por todos los demás al poner el dato, por ejemplo, en una variable compartida por todos ellos. En un ambiente distribuido esto no se puede ya que no hay variables compartidas por lo que sí o sí se debe mandar la información por medio de mensajes.
Con PMA existen dos opciones:
1) Utilizar un único canal desde donde todos los procesos pueden tomar el dato por lo que el emisor deberá enviar tanto mensajes como procesos receptores existan.
2) Utilizar un canal privado para comunicarse (enviar el mensaje) con cada uno de los procesos receptores.
El primer caso es mejor porque no hay demoras innecesarias ya que ni bien haya un mensaje en el canal este puede ser tomado por un receptor. En cambio, con la segunda opción un proceso que está listo para recibir el mensaje debe esperar a que el emisor le deposite el mensaje en su canal.
Con PMS no se puede usar un canal compartido porque todos son punto a punto, por lo tanto es más complejo e ineficiente. Es similar a la segunda opción con PMA ya que se debe enviar si o si un mensaje a cada proceso y además entre cada envío se debe esperar (SEND bloqueante) que el proceso lo haya recibido. Hay formas de optimizarlo como poner un buffer por cada proceso, y el emisor se lo envía a cada uno de esos buffer por si el proceso real está haciendo otro trabajo.

3)

a) De los problemas de los baños vistos en la teoría. ¿Cuál podría ser de exclusión mutua selectiva? ¿Por qué?

Opción 1: Un baño único para varones o mujeres (excluyente) sin límite de usuarios. Exclusión mutua selectiva porque tanto las mujeres o los hombres pueden seguir entrando después que uno de su clase obtuvo el acceso.
Opción 2: Un baño único para varones o mujeres (excluyente) con un número máximo de ocupantes simultáneos (que puede ser diferente para varones y mujeres). Es exclusión mutua general ya que cuando alguna de las clases que está ocupando un baño llega a su valor máximo de ocupantes simultáneos empieza a excluir a los de su propia clase, es decir a competir entre todos.
Opción 3: Dos baños utilizables simultáneamente por un número máximo de varones (K_1) y de mujeres (K_2), con una restricción adicional respecto que el total de usuarios debe ser menor que K_3 ($K_3 < K_1 + K_2$). Similar a la opción 2 hay un tope en las cantidades de cada clase por lo que en algún momento se empezaran a excluir entre todos.

a) ¿Por qué el problema de los filósofos es de exclusión mutua selectiva? Si en lugar de 5 filósofos fueran 3, ¿el problema seguiría siendo de exclusión mutua selectiva? ¿Por qué?

Un problema es de exclusión mutua selectiva cuando cada proceso compite por el acceso a los recursos no con todos los demás procesos sino con un subconjunto de ellos. El problema de los filósofos es de exclusión mutua selectiva ya que para comer un filósofo no compite con todos los demás sino que solo compite con sus adyacentes. En el caso de que fueran 3 filósofos y no 5 como en la definición del problema general, no sería de exclusión mutua selectiva ya que un proceso

compite con todos los demás procesos y no solo con un subconjunto de ellos, dado que el resto de los procesos son sus adyacentes.

b) **El problema de los filósofos resuelto de forma centralizada y sin posiciones fijas ¿es de exclusión mutua selectiva? ¿Por qué?**

Sin posiciones fijas se refiere a que cada filósofo solicita cubiertos y no necesariamente los de sus adyacentes, el coordinador o mozo se los da si es que hay dos cubiertos disponibles sin tener en cuenta vecinos, el problema no sería de exclusión mutua selectiva ya que competirían entre todos por poder acceder a los tenedores para poder comer.

c) El problema de los lectores-escritores es de exclusión mutua selectiva? ¿Porque?

En este caso, la exclusión mutua selectiva es entre clases de procesos ya que los procesos lectores como clase de proceso compiten con los escritores, esto se debe a que cuando un lector está accediendo a la BD otros lectores pueden acceder pero se excluye a los escritores.

d) Si en el problema de los lectores-escritores se acepta sólo 1 escritor o 1 lector en la BD, ¿tenemos un problema de exclusión mutua selectiva? ¿Por qué?

En este caso, al ser un único proceso lector el que compite el problema se convierte en un problema de exclusión mutua porque compiten entre todos los procesos por el acceso a un recurso compartido ya que no existen más lectores para acceder a la BD y lectores y escritores por definición se excluyen mutuamente.

4) **Analice que tipos de mecanismos de pasaje de mensajes son más adecuados para resolver problemas del tipo Cliente-Servidor, Pares que interactúan, filtro y productores- consumidores. Justificar.**

Pares que interactúan, filtros y productor-consumidor: Es más adecuado el uso PM ya que el flujo de comunicación es unidireccional. Además con RPC los procesos no pueden comunicarse directamente por lo que la comunicación debe ser implementada y con Rendezvous aunque los procesos si pueden comunicarse pero no pueden ejecutar una llamada seguida de una entrada de datos por lo que deberían definirse procesos asimétricos o utilizar procesos helpers.
Dependiendo de tipo de problema particular dependerá si es más adecuado PMA que provee buffering implícito y mayor grado de concurrencia o PMS que provee mayor sincronización.
Cliente-servidor: Es más adecuado el uso de RPC o Rendezvous ya que el problema requiere de un flujo de comunicación bidireccional el cliente solicita un servicio y el servidor responde a la solicitud, es decir que no solo el cliente le envía información para llevar a cabo la operación sino que el servidor le debe devolver los resultados de dicha ejecución, también no es necesario que el cliente siga procesando ya que antes de realizar otra tarea requerirá los resultados por parte del servidor.. Ambos mecanismos proveen este flujo de comunicación bidireccional con PM deberían utilizarse dos canales uno para las solicitudes y otro para las respuestas.

5) Describir la solución usando la criba de Eratóstenes al problema de hallar los números primos entre 2 y n. ¿Cómo termina el algoritmo? ¿Qué modificaría para que no termine de esa manera?

(Ver algoritmo aparte)

6) Sea el problema en el cual N procesos poseen inicialmente cada uno un valor V, y el objetivo es que todos conozcan cual es el máximo y cuál es el mínimo de todos los valores.

a) Plantee conceptualmente posibles soluciones con las siguientes arquitecturas de red: Centralizada, simétrica y anillo circular. No implementar.

En una arquitectura *centralizada* cada nodo o proceso le envía su dato al procesador central que es el encargado de ordenar los n valores y de reenviar el máximo y el mínimo valor a todos los demás procesos.
En la arquitectura *simétrica* hay un canal entre cada par de procesos y todos ejecutan el mismo algoritmo por lo tanto cada uno le envía su dato V a los n-1 procesos restantes y recibe n-1 valores. De este modo en paralelo todos están calculando el máximo y el mínimo.
Por último, en una arquitectura de *anillo* cada P[i] recibe mensajes de su predecesor P [i-1] y envía mensajes a su sucesor P [i+1]. Este esquema consta de 2 etapas; en la primera cada proceso recibe dos valores y los compara con su valor local, trasmitiendo un máximo local y un mínimo local a su sucesor. En la segunda etapa todos deben recibir la circulación del máximo y el mínimo global.

a) **Analice las soluciones anteriores desde el punto de vista del número de mensajes y la performance global del sistema.**

Para la primera solución con arquitectura *centralizada* se requieren 2(n-1) mensajes, n-1 para enviar los valores desde los procesos hacia el coordinador y n-1 enviados por el coordinador a los procesos con el máximo y el mínimo. Los mensajes al coordinador se envían casi al mismo tiempo por lo que solo el primer receive del coordinador demora mucho. Puede reducirse a n mensajes si el proceso central cuenta con una instrucción de broadcast. En cambio, con una arquitectura *simétrica* se envían n(n-1) mensajes que pueden transmitirse en paralelo si la red soporta transmisiones concurrentes pero el overhead de comunicación acota el speedup. Aunque es la más corta y sencilla utiliza la mayor cantidad de mensajes si no existe una instrucción de broadcast pero si la hay solo se necesitan n mensajes. En la arquitectura de *anillo* se requieren la misma cantidad que para la centralizada incluso si existe instrucción broadcast serán n mensajes. El último proceso debe esperar a que todos los otros (uno por vez) reciban un mensaje, hagan poco cómputo y envíen su resultado. Estos mensajes circulan 2 veces completas por el anillo por lo que la solución es lineal y lenta para este problema pero puede funcionar si cada proceso realiza mucho cómputo.
Tanto la arquitectura centralizada como en la arquitectura de anillo usan un número lineal de mensajes pero tienen distintos patrones de comunicación que llevan a distinta performance el patrón de pipeline circular de la arquitectura anillo hace que la solución sea más lenta.

7) **Defina el concepto de granularidad. ¿Qué relación existe entre la granularidad de programas y de procesadores?**

Se puede definir como *granularidad* a la relación que existe entre el procesamiento y la comunicación. En una *arquitectura de grano fino* existen muchos procesadores pero con poca capacidad de procesamiento por lo que son mejores para programas que requieran mucha comunicación. Entonces, los *programas de grano fino* son los apropiados para ejecutarse sobre esta arquitectura ya que se dividen en muchas tareas o procesos que requieren de poco procesamiento y mucha comunicación. Por otro lado, en una *arquitectura de grano grueso* se tienen pocos procesadores con mucha capacidad de procesamiento por lo que son más adecuados para *programas de grano grueso* en los cuales se tienen pocos procesos que realizan mucho procesamiento y requieren de menos comunicación.

8) **Dado el siguiente programa concurrente con memoria compartida, y suponiendo que todas las variables están inicializadas en 0 al empezar el programa, y que las instrucciones no son atómicas. Para cada una de las opciones indique verdadero o falso. En caso de ser verdadero indique el camino de ejecución para llegar a ese valor y de ser falso justifique claramente su respuesta.**

P1::	P2::	P3::
If (x = 0) then Y:= 4*x+2;	If (x > 0) then X := x+1;	X := x*8+x*2+1;

X:= y+2+x;		

a) **El valor de x al terminar el programa es 9.**

Verdadero.

a) **El valor de x al terminar el programa es 6.**

Verdadero.

b) **El valor de x al terminar el programa es 11.**

Falso. La única forma de que el valor de X sea 11 es que al ejecutar P3, X=1 y esto no puede darse porque P2 no puede alterar X porque X=0 y si P1 modificara el valor de X este seria 4.

c) **Y, siempre termina con alguno de los siguientes valores: 10 o 6 o 2 o 0.**

Verdadero.

Y=0: El único caso en el que Y sería 0, es que se ejecute P3 primero alterando el valor de X por lo que al ejecutarse P1 no realizaría ningún calculo.

Y=2: Siempre que asignación se realice antes de que los demás procesos alteren X (X=0). Si primero se ejecuta P1 completando la asignación de Y, el valor final de Y es 2 ya que ningún proceso vuelve a modificarla o modifico el valor de X.

Y=6: Si primero se ejecuta P2 terminando su ejecución porque x=0 y luego P1 evalúa el if y el control se pasa a P3 en ese instante, X=1 y al volver a realizar la asignación de Y en P1 el valor de Y=6.

Y=10: Se ejecuta P3 habilitando la asignación de P2 por lo que el valor de X=2 y luego se realiza la asignación de Y.

9) **¿En qué consiste la comunicación guardada (introducida por CSP) y cuál es su utilidad?**

Con frecuencia un proceso se quiere comunicar con más de un proceso, quizás por distintos canales y no sabe el orden en el cual los otros procesos podrían querer comunicarse con él. Es decir, podría querer recibir información desde diferentes canales y no querer quedar bloqueado si en algún canal no hay mensajes. Para poder comunicarse por distintos canales se utilizan sentencias de comunicación guardadas.

Las sentencias de comunicación guardada soportan comunicación no determinística: B; C->S;

- B puede omitirse y se asume true.
- B y C forman la guarda.
- La guarda tiene éxito si B es true y ejecutar C no causa demora.
- La guarda falla si B es falsa.
- La guarda se bloquea si B es true pero C no puede ejecutarse inmediatamente.

10) **¿Cuál es la utilidad de la técnica de passing the baton? ¿Qué relación encuentra con la técnica de passing the condition?**

La técnica de passing the baton es una técnica general para implementar sentencias await arbitrarias y para decidir cuál de los procesos es el próximo en seguir ejecutando (orden en el cual despertarlos). Ambas técnicas tienen la misma utilidad que es la de proveer un orden en la ejecución de procesos.

11) **Describa brevemente los mecanismos de comunicación y sincronización de Linda, Ada, MPI y Java.**

Linda utiliza memoria compartida como mecanismo de comunicación y pasaje de mensajes asincrónico como mecanismo de sincronización. En la memoria compartida hay tuplas que pueden ser activas (tareas) y pasivas (datos) pero el núcleo de LINDA es el espacio de tuplas compartido (TS) que puede verse como un único canal de comunicaciones compartido. Si bien hablamos de

memoria compartida el espacio de tuplas puede estar distribuido en una arquitectura multiprocesador.
Java provee concurrencia mediante el uso de threads, los threads comparten la zona de datos por lo que el mecanismo que utilizan para comunicarse es memoria compartida. Como método de sincronización java permite utilizar la palabra clave synchronized sobre un método lo que brinda sincronización y exclusión mutua. También permite la sincronización por condición con los métodos wait, notify y notifyAll, similares al "wait" y "signal" de los monitores, que deben usarse siempre dentro de un bloque synchronized. La semántica que utiliza el notify es de signal and continue.
Ada utiliza Rendezvous como mecanismo de sincronización y comunicación. En Ada un proceso realiza pedidos haciendo un call de un entry definido por otro proceso, este llamado bloquea al proceso llamador hasta que termina la ejecución del pedido. Para servir los llamados a sus entrys una task utiliza la sentencia accept que lo que hace es demorar a la tarea hasta que haya una invocación para el entry asociado a esa sentencia. Cuando recibe una invocación copia los parámetros y ejecuta la lista de sentencias correspondiente; al terminar copia los parámetros de salida y tanto el como el invocador pueden continuar su ejecución. Además, provee tres clases de sentencias select: wait selectivo que permite comunicación guardada, entry call condicional y entry call timed.
Wait selectivo: Select when B_1 => Sentencia accept; sentencias
Or…
Or when B_n => sentencia accept; sentencias
End select.
Se puede poner una sentencia else que es seleccionada si ninguna otra alternativa puede serlo, sentencia delay o sentencia terminate.
Entry call condicional: select entry call, sentencias
Else sentencias
End select
Solo se selecciona el entry call si puede ser ejecutado inmediatamente sino se selecciona el else.
Entry call timed: Select entry call; sentencias
Or delay; sentencias
End select
En este caso, el entry call se selecciona si puede ser ejecutado antes de que expire el intervalo delay.
MPI

12) Explique sintéticamente los 7 paradigmas de interacción entre procesos en programación distribuida. Ejemplifique.

Servidores replicados: Los servidores manejan (mediante múltiples instancias) recursos compartidos tales como dispositivos o archivos. Su uso tiene el propósito de incrementar la accesibilidad de datos o servicios donde cada servidor descentralizado interactúa con los demás para darles la sensación a los clientes de que existe un único servidor. Un ejemplo de servidores replicados es la resolución descentralizada al problema de los filósofos donde cada proceso mozo interactúa con otros para obtener los tenedores pero esto es transparente a los procesos filósofos.
Algoritmos de heartbeat: Los procesos periódicamente deben intercambiar información y para hacerlo ejecutan dos etapas; en la primera se expande enviando información (SEND a todos) y en la segunda se contrae adquieren información (RECEIVE de todos). Su uso más importante es paralelizar soluciones iterativas. Ejemplos de problemas que se pueden resolver son computación de grillas (labeling de imágenes) o autómatas celulares (el juego de la vida).
Algoritmos de pipeline: La información recorre una serie de procesos utilizando alguna forma de receive/send donde la salida de un proceso es la entrada del siguiente. Ejemplos son las redes de filtros o tratamiento de imágenes.

Algoritmos probe/echo: La interacción entre los procesos permite recorrer grafos o árboles (o estructuras dinámicas) diseminando y juntando información. Puede usarse para realizar un broadcast (sin spinning tree) o conocer la topología de una red cuando no se conocen de antemano la cantidad de nodos activos.
Algoritmos de Broadcast: Permiten alcanzar una información global en una arquitectura distribuida. Sirven para toma de decisiones descentralizadas y para resolver problemas de sincronización distribuida. Un ejemplo típico es la sincronización de relojes en un Sistema Distribuido de Tiempo Real.
Algoritmos Token passing: En muchos casos la arquitectura distribuida recibe una información global a través del viaje de tokens de control o datos. Permita realizar exclusión muta distribuida y la toma de decisiones distribuidas. Un ejemplo podría ser el de determinar la terminación de un proceso en una arquitectura distribuida cuando no puede decidirse localmente.
Manager/workers: Implementación distribuida del modelo de bag of tasks que consiste en un proceso controlador de datos y/o procesos y múltiples procesadores que acceden a él para poder obtener datos y/o tareas para ejecutarlos en forma distribuida.

13)

a) ¿Cuál es el objetivo de la programación paralela?

El objetivo principal de la programación paralela es reducir el tiempo de ejecución de algoritmos concurrentes o resolver problemas de tamaño muy grande o que requieran de una mayor precisión en los resultados en el mismo tiempo.

a) **Mencione las tres técnicas fundamentales de la computación científica. Ejemplifique.**

Entre las diferentes aplicaciones de cómputo científicas y modelos computacionales existen tres técnicas fundamentales:
(1) Computación de grillas (soluciones numéricas a PDE, imágenes). Dividen una región espacial en un conjunto de puntos.
(2) Computación de partículas. Modelos que simulan interacciones de partículas individuales como moléculas u objetos estelares.
(3) Computación de matrices. Sistemas de ecuaciones simultáneas.

b) Defina las métricas de speedup y eficiencia. ¿Cuál es el significado de cada una de ellas (que miden)? ¿Cuál es el rango de valores para cada una?

Ambas son métricas asociadas al procesamiento paralelo.
Speedup⇒ $S = T_s / T_p$: S es el cociente entre el tiempo de ejecución secuencial del algoritmo secuencial conocido más rápido (T_s) y el tiempo de ejecución paralelo del algoritmo elegido (T_p). El rango de valores de S va desde 0 a p, siendo p el número de procesadores. Mide cuánto más rápido es el algoritmo paralelo con respecto al algoritmo secuencial, es decir cuánto se gana por usar más procesadores.
Eficiencia⇒ E = S/P: Cociente entre speedup y número de procesadores. El valor está entre 0 y 1, dependiendo de la efectividad en el uso de los procesadores. Cuando es 1 corresponde al speedup perfecto. Mide la fracción de tiempo en que los procesadores son útiles para el cómputo, es decir cuánto estoy usando de los recursos disponibles.

c) ¿En qué consiste la ley de Amdahl?

La ley de Amdahl dice que para un dado problema existe un máximo speedup alcanzable independiente del número de procesadores. Esto significa que es el algoritmo el que decide la mejora de velocidad dependiendo de la cantidad de código no paralelizable y no del número de procesadores, llegando finalmente a un momento que no se puede paralelizar más el algoritmo.

14) Suponga que quiere ordenar N números enteros utilizando pasaje de mensajes con el siguiente algoritmo (odd/even Exchange sort): Hay n procesos P [1: n], con n par. Cada proceso ejecuta una serie de rondas. En las rondas impares, los procesos con número impar P[impar] intercambian valores con P [impar +1] si los números están desordenados. En las rondas pares, los procesos con numero par P[par] intercambian valores con P [par +1] si los números están desordenados (P [1] y P[n] no hacen nada en las rondas pares).

a) Determine cuantas rondas deben ejecutarse en el peor caso para ordenar los números.

Son necesarias n rondas en el peor de los casos que sería uno en el cual los valores se encuentran ordenados de mayor a menor porque requiere llevar el mayor valor n procesos más adelante y análogamente para el mínimo.

a) ¿Considere que es más adecuado para este caso, si pasaje de mensajes sincrónico o asincrónico? Justifique.

PMS es más adecuado en este caso porque los procesos deben sincronizar de a pares en cada ronda por lo que PMA no sería tan útil para la resolución de este problema ya que se necesitaría implementar una barrera simétrica para sincronizar los procesos de cada etapa.

b) Escriba un algoritmo paralelo para ordenar el arreglo a [1: n] en forma ascendente. ¿Cuántos mensajes se utilizan?

(Ver algoritmo aparte)

c) ¿Cómo modificaría el algoritmo del punto c para que termine tan rápido como el arreglo este ordenado? ¿Esto agrega overhead de mensajes? De ser así, ¿Cuánto?

Se puede usar un proceso coordinador al cual todos los procesos le envían en cada ronda si realizaron algún cambio o no. Si al recibir todos los mensajes el coordinador detecta que ninguno cambio nada les comunica que terminaron. Esto agrega overhead de mensajes ya que se envían mensajes al coordinador y desde el coordinador. Con n procesos tenemos un overhead de 2*n mensajes.

d) Modifique la respuesta dada en c para usar k procesos. Asuma que n es múltiplo de k.

(Ver algoritmo aparte)

15) Dado el siguiente programa concurrente con memoria compartida, tenga en cuenta que las instrucciones no son atómicas:

X:=4; y:=2; z:=3;

Co x:=x-z // z:=z*2 // y:=z+4 Oc

a) ¿Cuáles de las asignaciones dentro de la sentencia co cumplen con la propiedad de a lo sumo una vez? Justifique.

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

X:=X-Z Cumple la propiedad de a lo sumo una vez ya que posee una única referencia critica (Z) y X no es referenciada por otro proceso.
Z:=Z*2 Cumple la propiedad de a lo sumo una vez porque no tiene referencias críticas por lo tanto, puede ser leída por otro proceso.
Y:=Z+4 Cumple la propiedad de a lo sumo una vez ya que posee una referencia crítica e Y no es referenciada por ningún otro proceso.

b) **Indique los resultados posibles de la ejecución (No es necesario listarlos todos). Justifique.**

Cada tarea se ejecuta sin ninguna interrupción hasta que termina (Si una sentencia no es atómica se puede cortar su ejecución pero al cumplir ASV la ejecución no se ve afectada) y las llamamos T1, T2 y T3 respectivamente obtenemos el siguiente subconjunto de historias:

T1, T2, T3 => X=1, Z=6, y=10
T1, T3, T2 => X=1, Z=6, y=7
T2, T1, T3 => X=-2, Z=6, y=10
T2, T3, T1 => X=-2, Z=6, y=10
T3, T1, T2 => X=1, Z=6, y=7
T3, T2, T1 => X=-2, Z=6, y=7

El valor de Z es siempre el mismo ya que no posee ninguna referencia crítica. Los valores de X e Y se ven afectados por la ejecución de T2 ya que sus resultados dependen de la referencia que hacen a la variable Z que es modificada. Entonces, si T1 y T3 se ejecutan antes que T2 ambas usaran el valor inicial de Z que es 3 obteniendo los resultados X=1 e Y=7; ahora si T2 se ejecuta antes que las demas los resultados serán X=-2 e Y=10 y por último, tenemos los casos en que T2 se ejecuta en medio con T1 antes y T3 después o con T3 antes y T1 despues.

16) **Describa como es la ejecución de sentencias de alternativa e iteración que contienen comunicaciones guardadas.**

Sea la sentencia de alternativa con comunicación guardada de la forma:

If B1; comunicacion1 -> S1
□ B2; comunicacion2 -> S2
Fi

Primero, se evalúan las expresiones booleanas, B_i y la sentencia de comunicación.
•Si todas las guardas fallan (una guarda falla si B es false), el if termina sin efecto.
•Si al menos una guarda tiene éxito, se elige una de ellas (no determinísticamente).
•Si algunas guardas se bloquean, se espera hasta que alguna de ellas tenga éxito (B es true pero la ejecución de la sentencia de comunicación no puede ejecutarse inmediatamente).
Segundo, luego de elegir una guarda exitosa se ejecuta la sentencia de comunicación asociada.
Tercero, y último, se ejecutan las sentencias S relacionadas.
La ejecución de la iteración es similar realizando los pasos anteriores hasta que todas las guardas fallen.

17) **Utilice la técnica de passing the condition para implementar un semáforo fair usando monitores.**

(Ver algoritmo aparte)

18) **Analice conceptualmente la resolución de problemas con memoria compartida y memoria distribuida. Compare con respecto a facilidad de programación.**

La resolución de problemas con memoria compartida requiere el uso de exclusion mutua o sincronización por condición para evitar interferencias entre los procesos mientras que con memoria distribuida la información no es compartida si no que cada procesador tiene su propia memoria local y para poder compartir información se requiere del intercambio de mensajes evitando así problemas de inconsistencia. Por lo tanto, resulta mucho más fácil programar con memoria distribuida porque el programador puede olvidarse de la exclusion mutua y muchas veces de la necesidad de sincronización básica entre los procesos que es provista por los mecanismos de pasajes de mensajes. Igualmente la facilidad de programación con uno o con otro modelo también está dada por el problema particular que deba ser resuelto.

19) Broadcast atómico. Suponga que un proceso productor y n procesos consumidores comparten un buffer unitario. El productor deposita un mensaje 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.

a) Desarrolle una solución usando semáforos.

a) Suponga que el buffer tiene b slots. El productor puede depositar mensajes solo en slots vacíos y cada mensaje debe ser recibido por los n consumidores antes de que el slot sea reutilizado. Además, cada consumidor debe recibir los mensajes en el orden que fueron depositados. Extienda la respuesta dada en a para resolver este problema más general.

No lo está tomando porque es muy complejo.

20) Sea la siguiente solución propuesta al problema de alocación SJN:

```
Monitor SJN {
  Bool libre=true;
  Cond turno;
  Procedure request {
    If not libre wait (turno, tiempo);
    Libre = false;}
  Procedure release {
    Libre = true;
    Signal (turno) ;}
}
```

a) ¿Funciona correctamente con disciplina de señalización Signal and continue? Justifique.

Con S&C un proceso que es despertado para poder seguir ejecutando es pasado a la cola de ready en cuyo caso su orden de ejecución depende de la política que se utilice para ordenar los procesos en dicha cola. Puede ser que sea retrasado en esa cola permitiendo que otro proceso ejecute en el monitor antes que el por lo que podría no cumplirse el objetivo del SJN.

a) ¿Funciona correctamente con disciplina de señalización signal and wait? Justifique.

En cambio, con S&W se asegura que el proceso despertado es el próximo en ejecutar después de que el señalador ejecuta signal. Por lo tanto, SJN funcionaria correctamente de esta forma evitando que cualquier otro proceso listo para ejecutar le robe el acceso al proceso despertado.

21) Sea la siguiente solución propuesta al problema de alocacion LJN:

```
Monitor LJN {
Bool libre = true;
Cond turno;
Procedure request (int, tiempo){
        If (not libre) wait (turno, (Maxvalor - tiempo))
        Libre = false;}
Procedure release {
       Libre = true;
       Signal (turno)}}
```

a) ¿Funciona correctamente con disciplina de señalización Signal and continue? Justifique.

a) ¿Funciona correctamente con disciplina de señalización signal and wait? Justifique.

Pasa lo mismo que en SJN. La única manera de hacer que ambos funcionen correctamente con S&C es cambiar la implementación y usar passing the condition.
(Ver algoritmo aparte)

22)

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

RPC solo provee un mecanismo de comunicación y la sincronización debe ser implementada por el programador utilizando algún método adicional. En cambio, Rendezvous es tanto un mecanismo de comunicación como de sincronización.
En RPC la comunicación se realiza mediante la ejecución de un CALL a un procedimiento, este llamado crea un nuevo proceso para ejecutar lo solicitado. El llamador se demora hasta que el proceso ejecute la operación requerida y le devuelva los resultados. En Rendezvous la diferencia con RPC está en cómo son atendidos los pedidos, el CALL es atendido por un proceso existente, no por uno nuevo. Es decir, con RPC el proceso que debe atender el pedido no se encuentra activo sino que se activa para responder al llamado y luego termina; en cambio, con Rendezvous el proceso que debe atender los requerimientos se encuentra continuamente activo. Esto hace que RPC permita servir varios pedidos al mismo tiempo y que con Rendezvous solo puedan ser atendidos de a uno por vez.
Estos mecanismos son más adecuados para problemas con interacción del tipo cliente servidor donde la comunicación entre ellos debe ser bidireccional y sincrónica, el cliente solicita un servicio y el servidor le responde con lo solicitado ya que el cliente no debe realizar ninguna otra tarea hasta no obtener una respuesta del servidor.

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

Es necesario proveer sincronización dentro de los módulos porque los procesos pueden ejecutar concurrentemente. Estos mecanismos de sincronización pueden usarse tanto para el acceso a variables compartidas como para sincronizar interacciones entre los procesos si fuera necesario. En RPC existen dos modos de proveer sincronización y depende del modo en que se ejecutan los procesos dentro del módulo:
Si los procesos se *ejecutan por exclusión mutua*, solo hay un proceso activo a la vez dentro del módulo, el acceso a las variables compartidas tiene exclusión mutua implícita pero la sincronización por condición debe programarse. Pueden usarse sentencias await o variables condición.
Si los procesos *ejecutan concurrentemente* dentro del módulo debe implementarse tanto la exclusión mutua como la sincronización por condición para esto pueden usarse cualquiera de los mecanismos existentes semáforos, monitores, PM o Rendezvous.

c) ¿Qué elemento de la forma general de Rendezvous no se encuentra en ADA?

Rendezvous provee la posibilidad de asociar sentencias de scheduling y de poder usar los parámetros formales de la operación tanto en las sentencias de sincronización como en las sentencias de scheduling. Ada no provee estas posibilidades.

23) Resuelva con monitores el siguiente problema:

Tres clases de procesos comparten el acceso a una lista enlazada: searchers, inserters y deleters. Los searchers solo examinan la lista, y por lo tanto pueden ejecutar concurrentemente unos con otros. Los inserters agregan nuevos ítems al final de la lista; las inserciones deben ser mutuamente exclusivas para evitar insertar dos ítems casi al mismo tiempo. Sin embargo, un insert puede hacerse en paralelo con uno o más searchers. Por

último, los deleters remueven ítems de cualquier lugar de la lista a la vez, y el borrado también debe ser mutuamente excluyente con searchers e inserters.
(Ver algoritmo aparte)

24) **Suponga que el tiempo de ejecución de un algoritmo secuencial es de 1000 unidades de tiempo, de las cuales el 80% corresponden a código paralelizable. ¿Cuál es el límite en la mejora que puede obtenerse paralelizando el algoritmo?**

El límite de mejora se da teniendo 800 procesadores los cuales ejecutan una unidad de tiempo obteniendo un tiempo paralelo de 201 unidades de tiempo. El speedup mide la mejora de tiempo obtenida con un algoritmo paralelo comparándola con el tiempo secuencial. $S=T_s/T_p=1000/201=$ 4,97~5 aunque se utilicen más procesadores el mayor speedup alcanzable es el anterior cumpliéndose así la ley de Amdahl que dice que para un problema existe un límite en la paralización del mismo que no depende del número de procesadores sino que depende de la cantidad de código secuencial.

25) **Suponga los siguientes métodos de ordenación de menor a mayor para n valores (n par y potencia de dos), utilizando pasaje de mensajes:**
   1. Un pipeline de filtros. El primero hace un 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 procesador, mantiene el mínimo y pasa los otros al sucesor.
   2. Una red de procesos filtro como la del dibujo
   3. Odd/Even Exchange sort.

Asuma que cada proceso tiene almacenamiento local solo para dos valores (el próximo y el mantenido hasta ese momento).

a) **¿Cuántos procesos son necesarios en 1 y 2? Justifique.**

Para la alternativa del pipeline de filtros se necesitan n procesos ya que cada uno de ellos calculara un mínimo y pasara el resto de los valores. Por lo tanto, para terminar de procesar todos los números hacen falta tantos procesos como números se quieran ordenar.
Para una red de procesos filtro se necesitan n-1 procesos, ya que al ser un árbol binario con $\log_2 n$ niveles tenemos 2^n -1 nodos o procesos.

a) **¿Cuántos mensajes envía cada algoritmo para ordenar los valores? Justifique. NOTA: se pueden obviar en los cálculos los mensajes que se requieren para enviar los EOS.**

En un pipeline de filtros se puede definir el número de mensajes total requerido como $^{n}\sum_{i=1}$ (n-i). Cada proceso va reenviando un mensaje menos que la cantidad de mensajes que recibió. Además, a este resultado se le pueden sumar los mensajes de EOS, cada proceso envía un EOS obteniendo un total de n mensajes EOS.
En la red de filtros siendo n la cantidad de números a ordenar y $\log_2 n$ el total de niveles de la red tenemos que la cantidad de mensajes es $n \log_2 n$ a lo que podemos sumarle nuevamente los mensajes de EOS que son n.
Con odd/even Exchange sort si se tienen n procesos cada uno con 1 valor (como en el ejercicio sino seria números / procesos) la cantidad de mensajes requeridos es de n*(n-1).

b) **En cada caso ¿Cuáles mensajes pueden ser enviados en paralelo (asumiendo que existe el hardware apropiado) y cuales son enviados secuencialmente? Justifique. Ver más o menos porque me cuesta darme cuenta cuales pueden ser enviados en paralelo.**

En el pipeline de filtros en un instante determinado se podrán enviar en paralelo tantos mensajes como procesos tenga el pipeline ya que como cada uno de los procesos recibe, procesa y envía pueden todos estarse enviando los valores en un momento determinado.

En una red de filtros cada proceso envía de a un mensaje por vez ya que el funcionamiento interno de cada uno es igual a la de los filtros en un pipeline pero la diferencia está en la distribución e interacción entre ellos. Por lo tanto, se pueden enviar en paralelo tantos mensajes como procesos haya en el nivel en el que se esté dentro de la red.
Por último, en odd/even Exchange sort pueden enviarse en paralelo tantos mensajes como procesos se encuentren involucrados en una ronda ya que todos enviaran un valor.

c) **¿Cuál es el tiempo total de ejecución de cada algoritmo? Asuma que cada operación o envío de mensajes toma una unidad de tiempo. Justifique.**

Ejemplo del pipeline el tiempo total es el tiempo que lleva llenar el pipeline más el tiempo que lleva vaciarlo. Igualmente no me quedo del todo claro como calcular los tiempos.

26) **Resuelva con semáforos el problema del oso, las abejas y el tarro de miel: "Hay N abejas y un oso hambriento, que comparten un tarro de miel. El tarro inicialmente está vacío, y tiene una capacidad de H porciones. El oso duerme hasta que el tarro está lleno, luego come toda la miel y se vuelve a dormir. Cada abeja repetidamente produce una porción de miel y la pone en el tarro; la que llena el tarro despierta al oso".**

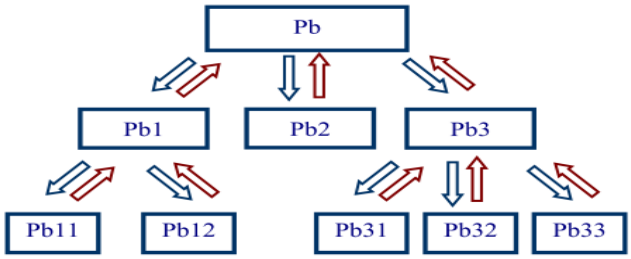
(Ver algoritmo aparte)

27) **¿Qué se entiende por arquitecturas de grano fino? ¿Son más adecuadas para programas con mucha o poca comunicación?**

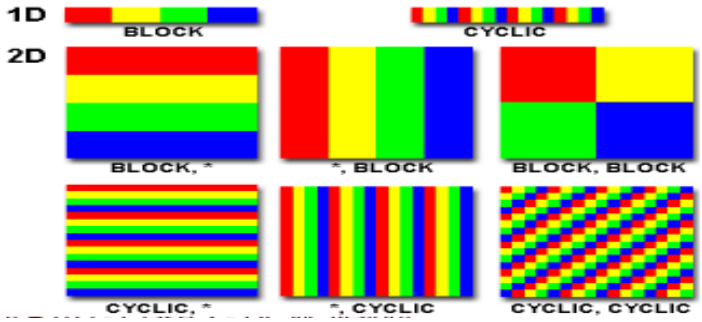
Una arquitectura de grano fino posee muchos procesadores pero con poca capacidad de procesamiento esto hace que las arquitecturas de grano fino sean más adecuadas para programas con mucha comunicación, es decir programas de grano fino donde existen muchos procesos que realizan poco cómputo pero donde se requiere de mucha comunicación.

28) **Explique sintéticamente los paradigmas de resolución de problemas concurrente. Ejemplifique.**

En el paralelismo recursivo el problema general se descomponerse en procesos recursivos que trabajan sobre partes del conjunto total de datos (Dividir y conquistar). Por ejemplo, el sorting by merging o juegos tipo ajedrez.



En el paralelismo iterativo un programa consta de un conjunto de procesos cada uno de los cuales tiene 1 o más loops. Luego, cada proceso es un programa iterativo. Los procesos cooperan para resolver un único problema, pueden trabajar independientemente, comunicarse y sincronizar por memoria compartida o pasaje de mensajes. Generalmente, el dominio de datos se divide entre los procesos siguiendo diferentes patrones. Un ejemplo es la multiplicación de matrices.



En el esquema de Productores- Consumidores, los procesos deben comunicarse y la comunicación fluye en una dirección por lo que generalmente se organizan en pipes donde la salida de uno será la entrada de su sucesor. Es decir, cada proceso es un filtro que consume la salida de su predecesor y produce una entrada para su sucesor.
En el esquema Cliente- Servidor, que es el predominante en los sistemas distribuidos, el servidor es un proceso que espera pedidos de servicio de múltiples clientes. Este tipo de esquema requiere de comunicación bidireccional y permite que clientes y servidores puedan ejecutarse en diferentes procesadores. La atención de clientes puede ser de a uno por vez (Rendezvous) o varios simultáneamente (RPC).
Con pares intractuantes los procesos resuelven partes de un problema, intercambian información mediante mensajes y a veces deben sincronizar para llegar a una solución. Este tipo de esquema permite mayor grado de asincronismo que el esquema de cliente-servidor. Existen diferentes configuraciones posibles: grilla, pipe circular, uno a uno, arbitraria.

29) Analice conceptualmente los modelos de mensajes sincrónicos y asincrónicos. Compárelos en términos de concurrencia y facilidad de programación.

Con PMS el send es bloqueante por lo que el emisor no puede realizar otras operaciones mientras espera que el mensaje sea recibido pero esta primitiva bloqueante provee un punto de sincronización. En términos de concurrencia PMA la maximiza con respecto a PMS ya que su send es no bloqueante por lo que un proceso puede seguir realizando tareas después de enviar un mensaje aunque este no haya sido recibido, para lograr esto PMA provee de buffering implícito. Además PMS tiene más riesgo de deadlock, que PMA por la semántica de la sentencia send, e implica muchas veces implementar procesos asimétrico o utilizar comunicación guardada. Por lo que desde el punto de vista de facilidad de programación es mejor PMA aunque la sincronización deba ser implementada por el programador en muchos casos. Igualmente la elección de un tipo u otro con respecto a la facilidad dependerá del tipo de problema que se requiera resolver ya que mucho problemas pueden por ejemplo ser resueltos con ambos mecanismos pero uno de ellos se adapta mejor haciendo más fácil la resolución.

30) **¿En qué consisten las arquitecturas SIMD y MIMD? ¿Para qué tipo de aplicaciones es más adecuada cada una?**

SIMD, Single intruction multiple data: Todos los procesadores ejecutan las mismas instrucciones pero sobre diferentes datos. Son para aplicaciones muy específicas de problemas regulares por ejemplo la multiplicación de matrices.
MIMD, multiple instruction multiple data: Cada procesador tiene su propio flujo de instrucciones y su propio conjunto de datos, es decir, cada uno ejecuta su propio programa. Pueden utilizar memoria compartida o memoria distribuida. Además pueden subclasificarse en MPMD donde cada procesador ejecuta su propio programa (PVN) y SPMD donde cada procesador ejecuta una copia del programa (MPI). Cualquier aplicación que no debe hacer lo mismo sobre distintos datos. Por ejemplo un pipeline o un master/worker o donde el trabajo es irregular, es decir que la cantidad de tiempo que

lleva cada tarea depende de los datos en sí y no del tamaño de los mismos como es la ordenación de vectores.

a) **Resuelva el problema de encontrar la topología de una red utilizando mensajes asincrónicos. Muestre con un ejemplo la evolución de la matriz de adyacencia para una red con al menos 7 nodos y de diámetro al menos 4.**
   (Ver algoritmo aparte)
b) **Compare conceptualmente con una solución utilizando PMS.**
   La solución con PMS dificultaría el algoritmo de forma tal que incrementaría la demora ya que se debe recibir en orden uno por uno los valores de los vecinos y si uno de ellos se retrasa no podemos continuar hasta que no recibamos el valor. Además de que se deberían usar algoritmos asimétricos de forma tal de evitar deadlock o utilizar comunicación guardada tanto para las sentencias de envió como para las de recepción. Por lo que en caso de resolver este algoritmo con PMS usando comunicación guardada sería más eficiente ya que solo recibiría de los canales en los que tiene datos sin necesidad de demorarse.

31) Defina sincronización entre procesos y mecanismos de sincronización.

La sincronización entre procesos puede definirse como el conocimiento de información acerca de otro proceso para coordinar actividades. Esta coordinación de actividades puede darse cuando se necesita acceder a valores compartidos, esperar resultados de otro proceso, etc.
Existen dos mecanismos de sincronización:
Por exclusión mutua: su objetivo es asegurar que solo un proceso tenga acceso a un recurso compartido en un determinado instante de tiempo. Si un programa tiene secciones críticas que pueden ser compartidas por más de un proceso, este mecanismo evita que varios procesos puedan encontrarse en la misma sección crítica en el mismo momento.
Por condición: Permite bloquear la ejecución de un proceso hasta que se cumpla una determinada condición.

32)

a) **¿Cuál (o cuáles) es el paradigma de interacción entre procesos más adecuado para resolver problemas del tipo "Juego de la vida"?**
   El paradigma que mejor se adecua es el heartbeat ya que permite enviar información a todos los vecinos y luego recopilar la información de todos ellos. Entonces una célula recopila la información de sus vecinos en la cual se basa su próximo cambio de estado.

b) **¿Considera que es conveniente utilizar mensajes sincrónicos o asincrónicos?**
   Es mejor la utilización de mensajes asincrónicos ya que evitan problemas de deadlock cuando se debe decidir qué proceso envía primero su información. Como en el algoritmo general de heartbeat todos los procesos primero envían su información y luego recopilan la información de sus vecinos se hace intuitivo el uso de PMA para su resolución evitando tener que realizar procesos asimétricos con demora innecesaria aunque la performance podría mejorar si se utiliza comunicación guardada para las sentencias de envío y recepción.

c) **¿Cuál es la arquitectura de hardware que se ajusta mejor? Justifique claramente sus respuestas.**
   Es conveniente una arquitectura de grano fino, con mucha capacidad para la comunicación y poca capacidad para el cómputo ya que este tipo de programas es de grano fino muchos procesos o tareas con poco computo que requieren de mucha comunicación. Una arquitectura en forma de grilla es una forma óptima para este tipo de problemas.

33) **Sea la siguiente solución al problema de multiplicación de matrices de n*n con P procesadores trabajando en paralelo.**

```
Process worker [w = 1 to P] {# strips en paralelo (p strips de n/P filas)
Int first = (w-1) * n/P + 1 # Primera fila del strip
Int last = first + n/P - 1; # Última fila del strip
For [i = first to last] {
  For [j = 1 to n] {
    C [i,j] = 0.0;
    for [k = 1 to n]  c[i,j] = c[i,j] + a[i,k]*b[k,j];}
}}
```

a) **Suponga n=128 y que cada procesador es capaz de ejecutar un proceso. ¿Cuántas asignaciones, sumas y productos se hacen secuencialmente (caso en el que P=1)?**
Si el algoritmo se ejecuta secuencialmente se tienen:
Asignaciones: $128^3 + 128^2 = 2097152 + 16384 = 2113536$
Sumas: $128^3 = 2097152$.
Productos: $128^3 = 2097152$.

b) **Si P1=...=P7 y los tiempos de asignación son 1, de suma 2 y de producto 3; si P8 es 4 veces más lento, ¿Cuánto tarda el proceso total? ¿Qué puede hacerse para mejorar el speedup? Modifique el código para lograr un mejor speedup.**
Si tenemos 8 procesos cada uno con un strip de 16 los cálculos de tiempo quedarían para cada proceso como:
Asignaciones: $128^2 * 16 + 128 * 16 = 262144 + 2048 = 264192$.
Sumas: $128^2 * 16 * 2 = 262144 * 2 = 524288$.
Productos: $128^2 * 16 * 3 = 262144 * 3 = 786932$.
Dando un total de tiempo por proceso de 1574912 unidades de tiempo. Pero P8 es 4 veces más lento entonces P8 requiere 1574912 * 4 = 6299648 unidades de tiempo haciendo que el *tiempo de ejecución total paralelo sea de 6299648*.
Usando los cálculos de a) obtenemos *el tiempo secuencial* con los tiempos de las operaciones.
Asignaciones: $128^3 + 128^2 = 2097152$
Sumas: $128^3 * 2 = 2097152 * 2 = 4194304$
Productos: $128^3 * 3 = 2097152 * 3 = 6291456$
Entonces el tiempo que requiere la ejecución secuencial es de 12599296 unidades de tiempo.
Con los procesos tal cual están definidos obtenemos un speedup de 12599296/6299648 = 2 pero que puede ser mejorado si realizamos un mejor balance de carga haciendo que P8 trabaje sobre un strip más pequeño. Dándole a P8 solo dos filas y los demás procesando 18 filas obtenemos los siguientes cálculos:

Asignaciones: $128^2 * 18 + 128 * 18 = 294912 + 2304 = 297216$.
Sumas: $128^2 * 18 * 2 = 294912 * 2 = 589824$.
Productos: $128^2 * 18 * 3 = 294912 * 3 = 884736$.
Entonces, P1…P7 ejecutan en 1771776 unidades de tiempo.
Asignaciones: $128^2 * 2 + 128 * 2 = 32768 + 256 = 33024$.
Sumas: $128^2 * 2 * 2 = 32768 * 2 = 65536$.
Productos: $128^2 * 2 * 3 = 32768 * 3 = 98304$.
P8 consume 196864 * 4 = 787456. Por lo que el tiempo paralelo de ejecución pasa a ser 1771776. Entonces, speedup = 12599296/1771776 =7,1 logrando una gran mejora con respecto al algoritmo secuencial gracias al balance de carga realizado para evitar retrasos por parte de P8.

c) **Lo mismo que en a) y b) con n=256.**
Son las mismas cuentas que antes pero reemplazando por 258 y con strips más grandes por lo que también se ve afectado el valor a cambiar con respecto a la cantidad de filas para balancear la carga.

34) **¿En qué consiste la utilización de relojes lógicos para resolver problemas de sincronización distribuida? Ejemplifique.**

Las acciones de comunicación, tanto send como receive, afectan la ejecución de otros procesos por lo tanto son eventos relevantes dentro de un programa distribuido y por su semántica debe existir un orden entre las acciones de comunicación. Básicamente cuando se envía un mensaje y luego este es recibido existe un orden entre estos eventos ya que el send se ejecuta antes que el receive por lo que puede imponerse un orden entre los eventos de todos los procesos. Sea un ejemplo el caso en que procesos solicitan acceso a un recurso, muchos de ellos podrían realizar la solicitud casi al mismo tiempo y el servidor podría recibir las solicitudes desordenadas por lo que no sabría quién fue el primero en solicitar el acceso y no tendría forma de responder a los pedidos en orden. Para solucionar este problema se podrían usar los relojes lógicos.

Para establecer un orden entre eventos es que se utilizan los relojes lógicos que se asocian mediante un timestamps a cada evento. Entonces, un reloj lógico es un contador que es incrementado cuando ocurre un evento dentro de un proceso ya que el reloj es local a cada proceso y se actualiza con la información distribuida del tiempo que va obteniendo en la recepción de mensajes. Este reloj lógico (rl) será actualizado de la siguiente forma dependiendo del evento que suceda:

Cuando el proceso realiza un SEND, setea el timestamp del mensaje al valor actual de rl y luego lo incrementa en 1. Cuando el proceso realiza un RECEIVE con un timestamp (ts), setea rl como max (rl, ts+1) y luego incrementa rl.

Con los relojes lógicos se puede imponer un orden parcial ya que podría haber dos mensajes con el mismo timestamp pero se puede obtener un ordenamiento total si existe una forma de identificar unívocamente a un proceso de forma tal que si ocurre un empate entre los timestamp primero ocurre el que proviene de un proceso con menor identificador.

35)

a) **Defina el concepto de sincronización barrier. ¿Cuál es su utilidad?**
Como muchos problemas pueden ser resueltos con algoritmos iterativos paralelos en los que cada iteración depende de los resultados de la iteración previa es necesario proveer sincronización entre los procesos al final de cada iteración, para realizar esto se utilizan las barreras. Las barreras son un punto de encuentro de todos los procesos luego de cada iteración, aquí se demoran los procesos hasta que todos hayan llegado a dicho punto y cuando lo hagan recién pueden continuar su ejecución.

b) **¿Qué es una barrera simétrica?**
Una barrera simétrica es un conjunto de barreras entre pares de procesos que utilizan sincronización barrier. En cada etapa los pares de procesos que interactúan van cambiando dependiendo a algún criterio establecido.

c) **Describa combining tree barrier y butterfly barrier. Marque ventajas y desventajas de cada una.**
En un combining tree barrier los procesos se organizan en forma de árbol y cumplen diferentes roles. Básicamente, los procesos envían el aviso de llegada a la barrera hacia arriba en el árbol y la señal de continuar cuando todos arribaron es enviada de arriba hacia

abajo. Esto hace que cada proceso deba combinar los resultados de sus hijos y luego se los pase a su padre.
Con butterfly barrier lo que se hace es en cada etapa diferente (son $\log_2$ n etapas) cada proceso sincroniza con uno distinto. Es decir, si s es la etapa cada proceso sincroniza con uno a distancia 2^{s-1}. Así al final de las $\log_2$ n etapas cada proceso habrá sincronizado direta o indirectamente con el resto de los procesos.

Combining tree barrier
Ventajas: Su implementación es más sencilla.
Desventajas: Los procesos no son simétricos ya que cada uno cumple diferentes roles, por lo que los nodos centrales realizaran más trabajo que los nodos hoja y la raíz.
Butterfly barrier
Ventajas: La implementación de sus procesos es simétrica ya que todos realizan la misma tarea en cada etapa que es la de sincronizar con un par a distancia 2^{s-1}.
Desventajas: Su implementación es más compleja que la del combining tree barrier y además cuando n no es par puede usarse un n próximo par para sustituir a los procesos perdidos en cada etapa; sin embargo esta implementación no es eficiente por lo que se usa una variante llamada Dissemination barrier.

36)

a) Suponga que la solución a un problema se paraleliza sobre p procesadores de dos maneras distintas. En un caso, el speedup (S) está dado por la función S=p-1 y en el otro por S=p/2. ¿Cuál de las dos soluciones se comportara más eficientemente al crecer la cantidad de procesadores? Justifique.
El que mejor se comportara con mayor número de procesadores es aquel cuya función de speedup es p-1 por definición de speedup su rango esta entre 0 y p entonces con esta función el speedup es más cercano a p. Si comparamos las eficiencias tenemos que:
E=S/p entonces tenemos p-1/p y (p/2)/p cuanto más grande sea p mayor eficiencia tendrá la primer solución ya que su valor será casi uno y la otra solución no alcanzara nunca una mayor eficiencia que la mitad.

b) **Ahora suponga S=1/p y S=1/p^2.**
Es más eficiente la solución con speedup 1/p su speedup siempre será mayor. Además su eficiencia será siempre mayor que la de la segunda solución, ambas a medida que crecen los procesadores van disminuyendo su eficiencia pero la función 1/p decrece más lentamente.

37) 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 de todos los elementos de una matriz pero difieren en la forma de llegar al resultado. P1 Genera le envía los 10000^2 valores a acumula para que este los sume mientras que en P2 Genera solo le envía 10000 valores ya que el mismo se encarga

de sumar toda una fila. Es decir, le envía a acumula la suma de todos los elemento de cada fila.

b) Analice desde el punto de vista del número de mensajes.
P1 realiza 10000^2 envíos de mensajes (uno por cada elemento de la matriz) mientras que P2 solo realiza 10000 (uno por cada fila de la matriz).

c) Analice desde el punto de vista de la granularidad de procesos.
Se puede decir que el proceso P2 es de grano más grueso que el proceso P1 ya que realiza mas procesamiento en Genera y esto hace que se necesite un menor número de comunicaciones con Acumula para llegar al resultado.

d) ¿Cuál de los programas le parece el más adecuado para ejecutar sobre una arquitectura tipo cluster PCs? Justifique.
Las arquitecturas del tipo cluster pueden considerarse arquitecturas de grano grueso ya que poseen muchos procesadores con mucha capacidad de cómputo pero con poca capacidad para la comunicación haciéndolos más apropiados para ejecutar programas de grano grueso que requieren de menos comunicación que capacidad de computo. Por lo tanto, P2 sería más apropiado para ejecutarse sobre este tipo de arquitecturas.

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);
  send canal (sum);   } }
```

```
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);      }
}
```

e) Responder las preguntas b, c y d para los siguientes programas.

En cuanto a cantidad de mensajes P1 realiza solo 10 envíos mientras que P2 realiza 10000 envíos. Estos envíos hacen de P1 un programa de grano grueso porque realiza más procesamiento y requiere de menos comunicación. Siendo la arquitectura de cluster una arquitectura de grano grueso por lo que es más apta para ejecutar el proceso P1 que requiere de menor comunicación y mayor capacidad de cómputo.

38) Dados los siguientes dos segmentos de código, indicar para cada ítem si son equivalentes o no. Justificar en cada caso (dar ejemplos si es necesario).

Segmento 1

```
...
int cant=1000;
DO  (cant < -10);  datos?(cant) → Sentencias1
  □ (cant > 10);  datos?(cant) → Sentencias2
  □ (INCOGNITA);  datos?(cant) → Sentencias3
END DO
...
```

Segmento 2

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

a) **INCOGNITA equivalente a: (cant = 0).**
No son equivalentes ya que el DO en segmento 1 terminaría para valores de cant (1...10) (-1…-10) mientras que en segmento 2 el if fallaría pero la iteración se seguiría realizando.

b) INCOGNITA equivalente a: (cant > -100)
Son equivalentes en segmento 1 con estas condiciones no hay ningún caso en el que todas las guardas fallen por lo tanto el DO no termina nunca, esto sería el mismo comportamiento que el while true con el IF.

c) INCOGNITA equivalente a: ((cant > 0) or (cant < 0))

No son equivalentes ya que el DO termina si cant=0 porque todas las guardas son false.

d) INCOGNITA equivalente a: ((cant > -10) and (cant < 10))

No son equivalente ya que para cant = 10 o cant=-10 segmento 1 terminaría su ejecución del DO porque todas las guardas serían falsas.

e) INCOGNITA equivalente a: ((cant >= -10) and (cant =< 10))

Serian equivalentes ya que se estarían eliminando en segmento 1 los valores puestos en d) que provocarían que el DO terminase.

39) **¿Qué significa el problema de interferencia en un programa concurrente? ¿Cómo puede evitarse?**

La interferencia se da cuando un proceso toma una acción que invalida alguna suposición hecha por otro proceso y esto se debe a que las acciones de los procesos en un programa concurrente pueden ser intercaladas. La interferencia se da por la realización de asignaciones en un proceso a variables compartidas que pueden afectar el comportamiento o invalidar un supuesto realizado por otro proceso. Por lo tanto, para evitar la interferencia entre procesos se usa la sincronización cuyo rol es restringir el número de historias (interleaving de sentencias) posibles solo a las deseables y para hacerlo existen dos mecanismos de sincronización: exclusion mutua o por condición.

40) En los protocolos de acceso a la SC vistos en teoría cada proceso ejecuta el mismo algoritmo. Una alternativa de resolver este problema es usando un proceso coordinador. En este caso, cada proceso SC[i] que quiere entrar a la SC le avisa al coordinador, y espera a que este le dé permiso. Al terminar de ejecutar su SC el proceso le avisa al coordinador. Desarrolle los protocolos de acceso para los procesos y el coordinador usando variables compartidas (no tenga en cuenta la propiedad de eventual entrada).

41) Clasificación de las arquitecturas multiprocesador:

a) Según el mecanismo de control. Describa.

Esta clasificación se basa en el modo en que las instrucciones son ejecutadas sobre los datos y hay 4 variantes.

SISD, Single instruction single data: Las instrucciones son ejecutadas en secuencia, de a una por ciclo de instrucción por lo tanto su ejecución en determinística. Básicamente la CPU ejecuta instrucciones sobre los datos, la memoria recibe los datos, los almacena y también brinda los datos. Usada en uniprocesadores.

SIMD, Single instruction multiple data: Todos los procesadores ejecutan las mismas instrucciones pero sobre diferentes datos.

MISD, multiple instruction single data: Todos los procesadores ejecutan diferentes flujos de instrucciones pero sobre los mismos datos.

MIMD, multiple instruction multiple data: Cada procesador tiene su propio flujo de instrucciones y su propio conjunto de datos, es decir, cada uno ejecuta su propio programa. Pueden utilizar memoria compartida o memoria distribuida. Además pueden subclasificarse en MPMD donde cada procesador ejecuta su propio programa (PVN) y SPMD donde cada procesador ejecuta una copia del programa (MPI).

b) Según la organización del espacio de direcciones. Describa.

Se clasifican en memoria compartida y memoria distribuida.

Con memoria compartida la interacción se realiza modificando los datos compartidos. Pueden existir problemas de consistencia. Dentro de esta clasificación se pueden usar dos esquemas UMA y NUMA con el primero la memoria física es compartida uniformemente por todo procesadores

mientras que con el esquema NUMA cada procesador tiene su propia memoria local pero esta puede ser accedida por los demás ya que el conjunto de todas las memorias locales forman una única memoria compartida.

Con memoria distribuida cada proceso tiene su propia memoria local que es usada únicamente por él y la interacción y el intercambio de información se realiza a través de pasaje de mensajes.

c) Según la red de interconexión. Describa.

Pueden clasificarse redes estáticas que son enlaces punto a punto y se utilizan para el pasaje de mensajes o redes dinámicas que se conectan mediante switches y enlaces de comunicación que normalmente son utilizadas para memoria compartida.

Dentro de las redes estáticas tenemos las diferentes topologías de red como anillo, estrella, árbol, hipercubo, etc. Y dentro de las redes dinámicas tenemos las redes multistage, árbol, bus, etc.

42) **Suponga que 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.

(Ver algoritmo aparte).

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

4 procesos (las esquinas) envían solo 2 mensajes por ronda. 4*2*(n-1)*2 = (n-1)*16 mensajes

(n-2) * 4 procesos (los bordes) envían 3 mensajes por ronda. (n-2)*2*3*(n-1)*2 = (n-1)(n-2)*12 mensajes.

(n-2)*(n-2) procesos envían 4 mensajes por ronda. $(n-2)^2$*4*(n-1)*2 = $(n-2)^2$*(n-1)*8.

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

No, no existe una manera de determinar cuándo un proceso obtuvo el mínimo o el máximo por lo que es necesario para asegurarse que todos los procesos tienen los valores globales de mínimo y máximo que se ejecuten las (n-1)*2.