Programacion Concurrente - Finales/Promocion

Tomatis, Vicens, Traberg

(Preguntas del resumen 1)

1. Defina programa concurrente, programa paralelo y programa distribuido.
   Respuesta:
   Un *programa concurrente* es aquel que tiene distintas tareas/procesos (que son programas secuenciales) que se ejecutan de forma simultanea.

   Programa paralelo
   Un programa concurrente en el cual los procesos se ejecutan en procesadores diferentes, y por lo tanto se ejecutan en paralelo.

   Programa distribuido
   Un programa en el cual los procesos se comunican con pasaje de mensajes, RPC o rendezvous. Usualmente en procesadores diferentes

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

   - *Propiedad del programa:*
     Una propiedad de un programa es un atributo que es cierto en cada historia posible del programa.
   - *Propiedad de seguridad:*
     Un programa nunca entra en un mal estado, por ejemplo donde las variables tienen un valor indeseado. Ej: exclusion mutua, ausencia de interferencia
   - *Propiedad de vida:*
     Un programa eventualmente entra en un buen estado, por ejemplo donde las variables tienen valores deseables. Ej: terminacion (ausencia de deadlock), fairness.

   b) Defina fairness. Relacionar dicho concepto con las políticas de scheduling
   Respuesta:
   Fairness: Trata de garantizar que todo proceso tenga la chance de ejecutarse. Las políticas de scheduling determinan el orden de ejecución (cuál será el próximo) y si garantizan la eventual entrada (según si es fairness débil o fuerte).

   c) Describa los distintos tipos de fairness
   Respuesta:
   Fairness incondicional: garantiza que eventualmente se van a ejecutar las acciones atomicas incondicionales
   Fairness debil: es incondicionalmente fair y toda accion atomica condicional es eventualmente ejecutada porque la solicitud es constante (es true y permanece en true).
   Fairness fuerte: es incondicionalmente fair y toda accion atomica condicional es eventualmente ejecutada porque la solicitud se hace de forma periodica (intercambia entre true y false).

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

   - **Exclusión mutua**: A lo sumo un proceso está en su SC
   - **Ausencia de deadlock**: Si 2 o más procesos tratan de entrar a sus SC, que al menos uno tenga éxito asegura la ausencia de deadlock.

- **No demora innecesaria**: Si un proceso trata de entrar a su SC y los otros no, el primero no está impedido de hacerlo.
- **Eventual entrada**: Un proceso que intenta entrar a su SC tiene posibilidades de hacerlo.

e) Cuáles son los defectos que presenta la sincronización por busy waiting? Diferencie esta situación respecto de los semáforos.
Respuesta:
Busy-waiting desperdicia tiempo en el procesador para solamente esperar a que la condicion se cumpla para poder acceder a la SC. Esto se resuelve en semáforos con las operaciones P y V que duermen al proceso o lo despiertan según se necesite.
Tiene tambien problemas con la separación de variables de sincronización con las de cómputo, esto se resuelve en semáforos con la variable de tipo semáforo, que abstraen el problema (de la resaca).
Otro de los problemas es que esta chequeando por una variable compartida que puede estar desactualizada en cache.

f) Explique la semántica de la instrucción de grano grueso AWAIT y su relación con las instrucciones Test & Set o Fetch & Add
Respuesta:
La instrucción AWAIT implica una condición booleana de demora y una serie de instrucciones a ejecutarse cuando la condición sea True. Al igual que TS y FA, es una acción atómica para lidiar con variables de manera exclusiva, solo que AWAIT consulta un valor para dar paso a instrucciones, mientras que TS y FA modifican los valores de sus variables, chequeando el valor de un booleano y cambiandolo a true si es false, pero devolviendo siempre el valor original (TS); e incrementando en 1 el valor actual devolviendo el valor viejo (FA). TS o FA se pueden usar para implementar sentencias await arbitrarias Ej:

Spin locks

```
bool lock=false;
process SC[i=1 to n]{
    while (true) {
        while (TS(lock)) skip ; --AWAIT
        sección crítica;
        lock = false;
        sección no crítica;
    }
}.
```

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.
Respuesta:
El problema general de la alocación de recursos se basa en que varios procesos compiten por recursos, para esto cada uno solicita ciertos recursos de un tipo, espera a que estén libres, los adquiere y luego los libera cuando termina de usarlos.

El algoritmo SJN trabaja muy bien con procesos breves que, de no darseles rápido su lugar en el recurso, se desprecia mucho tiempo en espera. Pero su principal problema es la ausencia de fairness. Esto se puede solucionar introduciendo la variante de ageing, donde a cada proceso se le asigna una edad que va incrementando por unidad de tiempo, esta edad es proporcional a la prioridad de entrada. Así, se da prioridad a los procesos breves pero se garantiza la eventual entrada de todos.

4. ¿En qué consiste la técnica de passing the baton? Aplicar este concepto a la resolución del problema de lectores y escritores.
Respuesta:
La técnica de Passing the Baton pretende persistir el orden solicitud de los procesos para el uso de un procesador, de manera tal que si hay alguno disponible, se le asigne uno al solicitante, mientras que, de no haber ninguno, se guarde constancia de este pedido para llamarlo, eventualmente, respecto a su orden de llegada, cuando un recurso se libere.

Aplicado a LyE:
Problema: dos clases de procesos (lectores y escritores) comparten una Base de Datos. El acceso de los escritores debe ser exclusivo para evitar interferencia entre transacciones. Los lectores pueden ejecutar concurrentemente entre ellos si no hay escritores actualizando.

Compartidas

```
int nr = 0, nw = 0, dr = 0, dw = 0;
sem e = 1, r = 0, w = 0:
```

Lector

```
process Lector [i = 1 to M]
{ while(true)
    { P(e);
    if (nw > 0)
        {dr = dr+1; V(e); P(r); }
    nr = nr + 1;
    if (dr > 0)
        {dr = dr - 1; V(r); }
    else
        V(e);
    lee la BD;
    P(e);
    nr = nr - 1;
    if (nr == 0 and dw > 0)
        {dw = dw - 1; V(w); }
    else
        V(e);
    }
}
```

Escritor

```
process Escritor [j = 1 to N]
{ while(true)
    { P(e);
    if (nr > 0 or nw > 0)
        {dw=dw+1; V(e); P(w);}
    nw = nw + 1;
    V(e);
    escribe la BD;
    P(e);
    nw = nw - 1;
    if (dr > 0)
        {dr = dr - 1; V(r); }
    elseif (dw > 0)
        {dw = dw - 1; V(w); }
    else
        V(e);
    }
}
```

5. Explique el concepto de broadcast y sus dificultades de implementación en un ambiente distribuido, con pasaje de mensajes sincrónico y asincrónico.
Respuesta:
El concepto de broadcast consiste en enviar concurrentemente un mismo mensaje a varios procesos. *Con PMA* existen dos opciones:

1) Utilizar un único canal general, donde el emisor envia tantos mensajes como procesos receptores existan. (si hay mas de una ronda puede causar problemas de tareas incorrectamente dadas) En este caso, no hay demora innecesaria, cuando se envía el mensaje, puede ser tomado por cualquiera de los receptores.

2) Utilizar un canal privado para enviar el mensaje a cada uno de los procesos receptores. (hay demora ya que un proceso que ya esta lista para trabajar no puedo empezar hasta que le entreguen a los anteriores)

Con PMS:
Al ser canales tipo link, se debe enviar un mensaje por cada receptor y se debe esperar a que se reciba, por el send bloqueante, puede ser optimizado con un buffer, para simplificar la sincronización

6. **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?
Respuesta: Exclusión mutua selectiva:
La exlusión mutua selectiva, se presenta cuando cada proceso compite contra un subconjunto de procesos (por un recurso). En lugar de competir contra todos.
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.
Sean 3 filosofos, como todos compiten contra todos los demás, deja de ser exclusion mutua selectiva, ya que todos los procesos son adyacentes.

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

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?
Respuesta:

En este caso, al ser un único proceso el que puede entrar a la S.C 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 ahora los lectores como conjunto no podrian acceder todos a la BD por lo que cada lector compite con todos.

7. 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.
Respuesta:
Cliente-Servidor: RPC y Rendevouz. Por su bidireccionalidad
Pares que interactúan: Pasaje de Mensajes. Por su asincronismo, facilidad de evitar deadlocks y mejoras de performance
Filtro: Pasaje de Mensajes. Por asincronismo: mejoras de performance
Productores-Consumidores: Pasaje de Mensajes. Lo mismo que la anterior

8. 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?
Respuesta: El problema requiere un algoritmo con un proceso diferenciado que inicialice el proceso, enviando al siguiente proceso todos los números impares a partir del 3. Luego, los demás procesos

recibirán un número, lo tomarán como propio, y luego recibirá el resto de los números y enviará al siguiente proceso sólo los números no divisibles por el propio del proceso.
El problema de esto es que los procesos no terminarán, pues esperan mensajes indefinidamente. Lo que se debe hacer es agregar un sentinela, por ejemplo -1, como valor final, que señale el fin del stream, y que el proceso, al recibir un valor, chequee esto y corte la ejecución de ser así.

Incorrecto

```
Process Criba[1]
{
  int p = 2;
  for [i = 3 to n by 2]
    Criba[2] ! (i);
}

Process Criba[i = 2 to L]
{
  int p, proximo;
  Criba[i-1] ? (p);
  do Criba[i-1] ? (proximo) →
    if ((proximo MOD p) <> 0 ) and (i
     ↪  < L)
      Criba[i+1] ! (proximo);
  od
}
```

Correcto

```
Process Criba[1]
{
  int p = 2;
  for [i = 3 to n by 2]
    Criba[2] ! (i);
  Criba[2]!(-1);
}

Process Criba[i = 2 to L]
{
  int p, proximo;
  bool seguir=true;
  Criba[i-1] ? (p);
  do (seguir)Criba[i-1] ? (proximo) →
    if ((proximo MOD p) <> 0 ) and (i
     ↪  < L)
      Criba[i+1] ! (proximo);
    if(proximo == -1)
      seguir=false;
    -- lo de abajo es por si L es
     ↪  menor a la cantidad de
    -- numeros primos entre 2 y n, y
     ↪  se quieren guardar
    -- el resto de numeros primos con
     ↪  el ultimo
    if (i==L and proximo!=-1 and
     ↪  (proximo MOD p)<>0)
      if (empty(colaUltimo))
        push(colaUltimo,proximo)
      for (primo in colaUltimo)
        if(proximo MOD primo==0)
            break
        if(primo == last(colaUltimo))
            push(colaUltimo,proximo)
  od
}
```

9. 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.

Respuesta:
Centralizada: N-1 procesos se comunican con un proceso central al cual le da información local y, luego de que todos lo hagan, el central se comunica con los procesos para darles la informacion final.
Simetrica: Cada uno de los N procesos se comunica con el resto, enviándole su valor, y cada uno de ellos procesa toda la información de manera local.
Anillo circular: Un primer proceso envía al siguiente su número, y los otros N-1 procesos reciben un valor del proceso anterior, lo procesa y le envía al siguiente el resultado y espera recibir el resultado final.

b) Analice las soluciones anteriores desde el punto de vista del número de mensajes y la performance global del sistema.
Respuesta:
Simetrica: N*(N-1) mensajes. Es la mas rapida de todas pero con mucho overhead de mensajes
Centralizada: 2(N-1) mensajes. Su performance es la intermedia entre circular y simetrica, bastante rapida con la posibilidad de agregar mas nodos centrales para evitar bottlenecks
Circular: 2N-1 mensajes. Es la mas lenta de todas ya que requiere un transporte de datos de a lo sumo n/2 pasos.

10. Defina el concepto de granularidad. ¿Qué relación existe entre la granularidad de programas y de procesadores?
Respuesta:
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 comunicacion entre tareas o procesos que requieren poco procesamiento (programa de grano fino).
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.

11. 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;

Respuesta:

a) El valor de x al terminar el programa es 9. ☑P2 completo - P1 condicion - P3 primera linea - P1 segunda linea - P3 segunda linea

b) El valor de x al terminar el programa es 6. ☑P1 completo - P3 primera linea - P2 completo - P3 segunda linea

c) El valor de x al terminar el programa es 11. ☒

d) Y, siempre termina con alguno de los siguientes valores: 10 o 6 o 2 o 0. ☒P1 condicion - P3 primera linea - P2 completo - P3 segunda linea - P1 segunda linea

12. ¿En qué consiste la comunicación guardada (introducida por CSP) y cuál es su utilidad?
Respuesta:
Con frecuencia un proceso quiere comunicarse 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 esto el do y el if de CSP usan comunicación guardada con guardas de la forma (condicion booleana); (comunicación) -> (sentencias a ejecutar)

13. ¿Cuál es la utilidad de la técnica de passing the baton? ¿Qué relación encuentra con la técnica de passing the condition?
Respuesta:
La técnica de passing the baton provee un orden en la ejecución de procesos para la entrada y salida de la seccion critica, passing the condition tiene el mismo objetivo, pero es implementado en monitores, aprovechando la exclusión mutua provista por estos.

14. Explique sintéticamente los 7 paradigmas de interacción entre procesos en programación distribuida. Ejemplifique.
Respuesta:

- Manager and workers: Es la implementación distribuida del modelo bag of tasks. Diversos procesos workers comparten una bolsa de tareas. Cada worker repetidamente toma una tarea de la bolsa, la ejecuta y posiblemente genere una (o más) tareas nuevas para depositar en la bolsa.
- Heartbeat: Los procesos, periódicamente, deben intercambiar información con mecanismos de tipo send/receive.
  La interacción entre vecinos, produce una especie de barrera de sincronización entre ellos. Impidiendo que, cada worker inicie su fase de actualización antes de que su vecino no finalice la suya.
- Pipeline: Los procesos, funcionan como filtros y se encuentran conectados entre sí; formando una colección de filtros. De esta manera, la información fluye por los procesos utilizando alguna forma de receive/send.
- Probes and echoes: La interacción entre procesos permite recorrer estructuras dinámicas, diseminando y recolectando información.
  Una primitiva probe es un mensaje enviado por un nodo, a todos sus sucesores. Mientras que, una primitiva echo es su respectiva respuesta. Este paradigma, es el análogo al algoritmo Depth-first Search (DFS).
- Broadcasts: Los procesos suelen enviar mensajes a todos los demás, de forma paralela, mediante una primitiva broadcast.
  Dicha primitiva, no se asume atómica, por lo cual los procesos pueden recibir los mensajes en distinto orden al cual fueron enviados.
- Token passing: Lo procesos se pasan, entre sí, un token que garantiza el acceso exclusivo a un recurso compartido en ambientes distribuidos.
- Servidores replicados: Los procesos servidores gestionan el acceso a un recurso. Al existir N instancias del recurso, se replicará un servidor por cada una.

15. **a)** ¿Cuál es el objetivo de la programación paralela?
Respuesta:
El objetivo de la programación paralela es disminuir el tiempo total de ejecución de un algoritmo, dividiendo el trabajo en tareas menores que puedan ser ejecutadas por distintos procesos en simultaneo.

b) Mencione las tres técnicas fundamentales de la computación científica. Ejemplifique.
Respuesta:

- Computación de grillas: Dividen una región espacial en un conjunto de puntos, se usan para resolver Ecuación en derivadas parciales. Ej: Procesamiento de imágenes
- Computación de partículas: Modelos que simulan interacciones de partículas individuales como moléculas. Ej: Simulación de fluidos
- Computación de matrices. Sistemas de ecuaciones simultáneas.

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 para cada una?
Respuesta:
El speedup es el cociente entre el tiempo más rápido de ejecución del algoritmo en forma secuencial (T_s) y el tiempo de ejecución paralelo del algoritmo elegido (T_p). $S = T_s/T_p$ El Speedup mide cuánto más rápido es la ejecución en forma paralela respecto de la secuencial. Su rango de valores es, en general, entre 0 y S_{optimo}. Si el speedup es igual al speedup optimo se dice que es lineal/perfecto, si esta por debajo se le dice sublineal y si esta por arriba (ocasiones muy raras donde depende la localidad de la memoria u otros problemas) se le dice superlineal.

La eficiencia es el cociente entre el speedup y el speedup óptimo. $E = S/S_{optimo}$;
$S_{optimo} = \sum_{i=0}^{P} \frac{PotenciaCalculo(mejor)}{PotenciaCalculo(i)}$
Mide la fracción de tiempo en que los procesadores son útiles para el cómputo. El valor está entre 0 y 1, dependiendo de la efectividad de uso de los procesadores. Cuando es 1 corresponde al speedup perfecto.

d) ¿En qué consiste la ley de Amdahl?
Respuesta:
La ley de Amdahl postula que; para todo algoritmo existe un speedup máximo alcanzable (límite), independiente del número de procesadores. Ese valor dependerá de la cantidad de código paralelizable. Si el 80% de un programa es paralelizable:

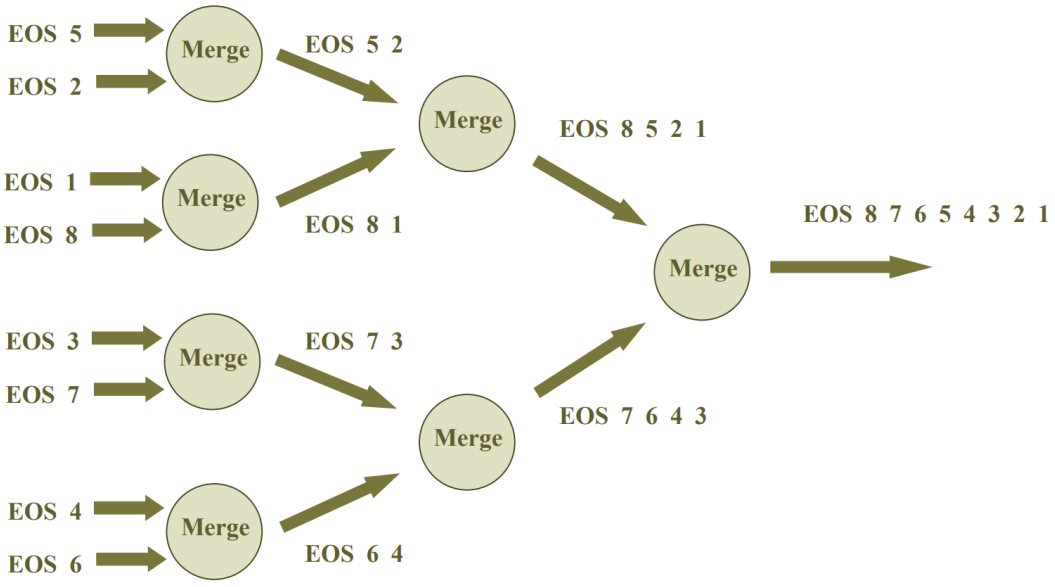
$$Limite = 1/(1 - Paralelizable) = 1/(1 - 0,8) = 1/0,2 = 5$$

16. 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.
Respuesta:
En el peor caso, para ordenar los números se necesitarán K rondas. En cuanto a la cantidad de mensajes, la fórmula (del matemático Relderf) es N*(K-1), donde N es la cantidad de números a ordenar y K es la cantidad de procesos, determina el máximo de mensajes enviados para ordenar el arreglo en el peor de los casos.
Una forma de entender la formula es que hay k rondas con k procesos que intercambian n/k mensajes, pero aunque el numero de procesos sea par o impar en promedio por ronda no se pueden hacer n/k intercambios por las esquinas (en par ambas esquinas cada 2 rondas, en impar todas las rondas una esquina diferente), por lo que la cantidad de mensajes seria: $k * k * n/k - k * n/k = nk - n = n(k - 1)$

b) ¿Considere qué es más adecuado para este caso, si pasaje de mensajes sincrónico o asincrónico? Justifique.
Respuesta:
Para este caso es más adecuado pasaje de mensaje sincrónicos, porque las rondas son consecutivas pero los mensajes entre procesos en cada ronda serán simultaneos, y cada proceso estará interactuando, como máximo, con un proceso a la vez. El uso de pasaje de mensajes asincrónicos podría introducir, por error, un defazaje en el procesamiento de la información y alterar la lógica implicita del algoritmo (en la cual, si el número más grande de un proceso es menor que el más chico del siguiente, ambos ya están ordenados en conjunto).

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



La cantidad de mensajes es $N * h + (2^h - 1) * 2$ siendo h la altura del arbol.

d) ¿Cómo modificaría el algoritmo del punto c para que termina tan rápido como el arreglo este ordenado? ¿Esto agrega overhead de mensajes? De ser así, ¿Cuánto?
Respuesta:
Se podría implementar un proceso coordinador para lograr la siguiente interacción:. Al final de cada ronda, los procesos le indican al coordinador si efectuaron cambios en su colección de números Si el coordinador, para dos rondas consecutivas(una de cada paridad), no recibe ninguna notificación de cambios entonces; da por concluido el trabajo y notifica a los procesos. Agrega overhead de mensajes: Por cada ronda, los procesos involucrados le deben enviar 1 aviso de que hubo cambios al coordinador y este debe responder con un mensaje que indique parada o continuación. En total 2k*la cantidad de rondas que se realizen

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

```
Process worker[i = 1..k]
    int largest = n/k, smallest = 1,
    a[1:k], dato;
    # Ordeno la porcion del arreglo
    # del proceso actual.
```

```
    for(ronda = 1; ronda <= k; ronda++)
        if (i mod 2 == ronda mod 2)
            # Misma paridad (proceso+ronda par o proceso+ronda impar)
            if (i != k)
                proc[i+1]!(a[largest]);
                proc[i+1]?(dato);
                while (a[largest] > dato)
                    # Inserto dato ordenado, pisando a[largest].
                    proc[i+1]!(a[largest]);
                    proc[i+1]?(dato);
                end;
            end;
        else
            if (i != 1)
                proc[i-1]?(dato);
                proc[i-1]!(a[smallest]);
                while (a[smallest] < dato)
                    # inserto dato ordenado, pisando a[smallest].
                    proc[i-1]?(dato);
                    proc[i-1]!(a[smallest]);
                end;
            end;
        end;
    end;
End.
```

17. 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
Respuesta:
X:=X-Z porque la única referencia crítica es Z, que no tiene referencias críticas.
Z:=Z*2 porque no posee referencias críticas.
Y:=Z+4 por la misma razón que la primera.

b) Indique los resultados posibles de la ejecución (No es necesario listarlos todos). Justifique.
Respuesta:
1) X=-2; Z=6; Y=10 2) X=1; Z=6; Y=7 3) X=1; Z=6; Y=10

18. Describa como es la ejecución de sentencias de alternativa e iteración que contienen comunicaciones guardadas.
Respuesta:
· DO: El proceso entra en un bloque del cual evalúa la veracidad de las guardas de las opciones disponibles (la ausencia de condición booleana en una guarda implica veracidad (True)) y toma, de

forma no determinística, una para ejecutar. Luego, se vuelve a evaluar las guardas de las opciones y se vuelve a tomar de manera no determinística otra opción con guarda verdadera para ejecutar. Cuando no hay mensajes para recibir y las condiciones booleanas de las guardas de uno o más opciones son verdaderas, la opción se bloquea y el DO se queda a la espera de que una de la opciones se active para elegir nuevamente una de forma de no determinística . Cuando no haya ninguna guarda verdadera, se sale del bucle.
· IF: Funciona como el DO, con la diferencia de que luego de una iteración, se sale del bucle, sea exitoso o no.

19. Utilice la técnica de passing the condition para implementar un semáforo fair usando monitores.
Respuesta:

Monitor admin

REVISAR!

20. Analice conceptualmente la resolución de problemas con memoria compartida y memoria distribuida. Compare con respecto a facilidad de programación.
Respuesta:
En memoria distribuida la exclusión mutua es implícita, mientras que en memoria compartida es necesario escribirla y esto puede conllevar muchos errores humanos. En el caso de semáforos, la lectura se vuelve más engorrosa, cuando hay muchas variables de exclusión mutua anidadas, lo cual no sucede en pasaje de mensajes.

21. 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.
Respuesta:
No, debido a que, al despertar al proceso dormido primero en la fila, éste pasa a competir por usar el monitor, y si otro proceso pide el uso del recurso en ese mismo momento, ambos competirán y podría no respetarse el orden

b) ¿Funciona correctamente con disciplina de señalización signal and wait? Justifique.
Respuesta:
Sí, ya que luego del signal, el proceso que ocupa el monitor se sale de este, e inmediatamente el proceso despetado toma control del monitor y lockea la variable booleana, haciendo que otros procesos solicitantes se duerman en la variable condición.

22. 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.
Respuesta:
Igual que la 22.

b) ¿Funciona correctamente con disciplina de señalización signal and wait? Justifique.
Respuesta:
Igual que la 22.

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

Respuesta:
· RPC: Los programas se descomponen en módulos (con procesos y procedures), que pueden residir en espacios de direcciones distintos. Los procesos de un módulo pueden compartir variables y llamar a procedures de ese módulo. Un proceso en un módulo puede comunicarse con procesos de otro módulo sólo invocando procedimientos exportados por éste.
Son más adecuados para problemas de tipo manager and workers, ya que potencia la concurrencia.
· Rendezvous: Como con RPC, un proceso cliente invoca una operación por medio de un call, pero esta operación es servida por un proceso existente en lugar de por uno nuevo. Un proceso servidor usa una sentencia de entrada para esperar por un call y actuar.
Las operaciones se atienden una por vez más que concurrentemente.
Son más adecuados para problemas de tipo cliente servidor ya que facilitan la interacción, por su canal bidireccional.

b) ¿Por qué es necesario proveer sincronización dentro de los módulos RPC? ¿Cómo puede realizarse esta sincronización?
Respuesta:
Por sí mismo, RPC es solo un mecanismo de comunicación. Al tener mas de un proceso que pueden utilizar variables compartidas es necesario sincronizar el uso de las mismas ya sea por exclusion mutua o por condicion. La mejor tecnica que aproveche la concurrencia seria semaforos ya que permitiria ejecutar cada proceso y al mismo tiempo evitar la interferencia entre los mismos.

c) ¿Qué elemento de la forma general de Rendezvous no se encuentra en ADA?
Respuesta:
El select de Ada, a diferencia del rendezvous general no tiene expresiones de scheduling ni puede referenciar parámetros formales de llamadas a procedimientos.

24. 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 puede 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.
Respuesta:
Asumiendo signal and continue

```
Process searchers
{
    Acceso.entrarSearcher()
    CosasLista()
    Acceso.salir(0)
}

Process inserter
{
    Acceso.entrarInserter()
    CosasLista()
    Acceso.salir(1)
}

Process deleter
{
    Acceso.entrarDeleter()
    CosasLista()
    Acceso.salir(2)
}

Monitor Acceso {
    cond searchers;
    cond deleters;
    cond inserters;
    int adentro[3]

    procedure entrarSearcher(){
        while adentro[2] > 0 {
            wait(searchers);
        }
        adentro[0]++
    }

    proceduce entrarInserter(){
        while adentro[1] > 0 or adentro[2] > 0 {
            wait(inserters);
        }
        adentro[1]++
    }

    procedure entrarDeleter(){
        while adentro[0] > 0 or adentro[1] > 0 or adentro[2] > 0 {
            wait(deleters);
        }
        adentro[2]++
    }

```

```
49      procedure salir(int tipo){
50          adentro[tipo]--
51          if not empty(deleters)
52              signal_all(deleters)
53          else{
54              if not empty(inserters)
55                  signal_all(inserters)
56              else{
57                  if not empty(searchers)
58                      signal_all(searchers)
59              }
60          }
61      }
62  }
```

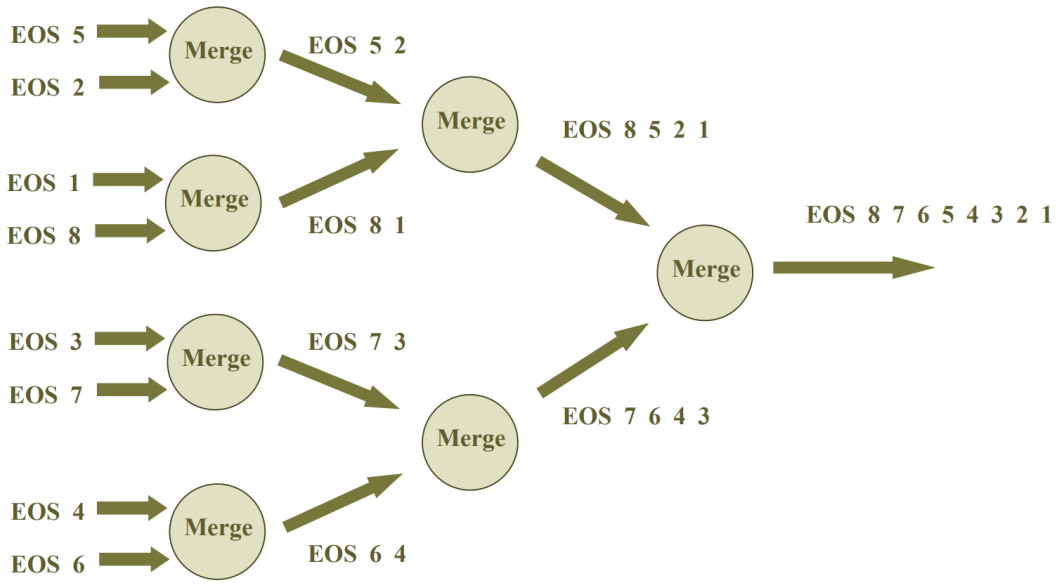
25. Suponga que el tiempo de ejecución de un algoritmo secuencial es de 1000 unidades de tiempo, de las cuales el 80% corresponden a código paralelizable ¿Cuál es el límite en la mejora que puede obtenerse paralelizando el algoritmo?
Respuesta:

$$1/(1-0.8) = 1/0.2 = 5$$

El algoritmo puede ser hasta 5 veces más rápido paralelizando ese 80%.

26. Suponga los siguientes métodos de ordenación de menor a mayor para n valores (n par y potencia de dos), utilizando pasaje de mensajes:

- 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.
- Una red de procesos filtro como la del dibujo



- 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.
Respuesta:
Para el 1 son necesarios N, porque ningún proceso puede ser reutilizado para calcular valores siguientes luego de terminar.
Para el 2 son necesarios N-1 procesos.

b) ¿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.
Respuesta:
El 1 es $\sum_{i=1}^{n} i = n(n+1)/2$, porque cada proceso, salvo el último, le envía un mensaje al siguiente.
El 2 es $N * h + (2^h - 1) * 2$ siendo h la altura del arbol, la misma razon que el 1. Asumiendo que cada proceso tiene los 2 valores
El 3 es N*(K-1) para el peor caso donde el array está en orden inverso

c) En cada caso ¿Cuáles mensajes pueden ser enviados en paralelo (asumiendo que existe el hardware apropiado) y cuales son enviados secuencialmente? Justifique.
Respuesta:
En el 1, los mensajes del stream de numeros pueden paralelizarse ya que cuando el proceso b, envía el mensaje 1, el proceso a puede enviar el mensaje 2
El 2 se pueden mandar en paralelo los mensajes de los hijos a los padres pero cada padre debe esperar a los mensajes de los hijos
El 3 se pueden mandar en paralelo los mensajes entre par/impar y el siguiente, pero el intercambio de numeros se realiza de forma secuencial tambien respetando las rondas.

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

27. 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".
Respuesta:

HACER!!

28. ¿Qué se entiende por arquitecturas de grano fino? ¿Son más adecuadas para programas con mucha o poca comunicación?
Respuesta:
Son arquitecturas con poco poder de procesamiento pero muchas unidades de procesamiento, apropiadas para programas con más comunicación por su bajo poder de cómputo.

29. Explique sintéticamente los paradigmas de resolución de problemas concurrente. Ejemplifique.
Respuesta:

- Paralelismo iterativo: Se usa cuando se poseen varios procesos idénticos, que se comunican y sincronizan por memoria compartida o pasaje de mensajes. Ej: Multiplicación de matrices

- Paralelismo recursivo: En el paralelismo recursivo el problema general se descomponerse en procesos recursivos que trabajan sobre partes del conjunto total de datos (Dividir y conquistar). Ej: sorting, scheduling.
- Productores Consumidores: 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.
- Cliente Servidor: Un cliente requiere un servicio y espera a recibir una respuesta. El servidor espera requests y las responde, el servidor puede manejar 1 o varios requests a la vez si es mutihilo.
- Pares que interactuan: 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.

30. Analice conceptualmente los modelos de mensajes sincrónicos y asincrónicos. Compárelos en términos de concurrencia y facilidad de programación.
Respuesta:
En términos de concurrencia, PMA no es bloqueante, lo que le permite enviar mensajes y seguir haciendo otras tareas sin necesidad de esperar la recepción. Mientras que en PMS el envío de mensajes es bloqueante e implica la recepción para seguir.
La programación en PMS es más fácil porque no se deben declarar explícitamente los links, mientras que en PMA los canales deben ser escritos, en PMS es más propenso a deadlocks por errores humanos.
En PMA los canales son de tipo mailbox (a su vez pueden simular el resto de tipos) lo que permite tener una cola implicita de mensajes, a diferencia de PMS que los canales al ser de tipo link no permite haber colas de mensajes.

31. ¿En qué consisten las arquitecturas SIMD y MIMD? ¿Para qué tipo de aplicaciones es más adecuada cada una?
Respuesta:
SIMD: Single Instruction Multiple Data

- Conjunto de procesadores simples, idénticos, que ejecutan la misma instrucción sobre distintos datos.
- El host hace broadcast de la instrucción. Ejecución sincrónica y determinística.
- Adecuados para aplicaciones con alto grado de regularidad, (por ejemplo procesamiento de imágenes).

MIMD: Multiple Instruction Multiple Data

- Cada procesador tiene su propio flujo de instrucciones y de datos ejecutando su propio programa.
- Pueden ser con memoria compartida o distribuida.
Ej: 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.

1) 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.
Respuesta:

```
## con diametro
diametro=D
for 1 to diametro
    envio mi matriz a mis vecinos
    recibo las matrices de mis vecinos
    actualizo mi matríz

# sin diametro
termine=false
while (not termine)
    envio mi matriz a mis vecinos que no terminaron
    recibo las matrices de mis vecinos que no terminaron
    actualizo mi matríz
    termine=checkeo si hay al menos algun 1 en cada fila de la matriz
envio la matriz completada con el flag de no me molesten o termine
recibe mensajes para evitar dejarlos en el aire
```

2) Compare conceptualmente con una solución utilizando PMS.
Respuesta:
En PMS es muy difícil de hacerlo o casi imposible. Podrían dividirse en pares e impares, y en base a su ID reaccionar de una forma u otra, pero meh.

32. Defina sincronización entre procesos y mecanismos de sincronización.
Respuesta:
La sincronización es la posesión de información acerca de otro proceso para coordinar actividades.

- Exclusión mutua: Asegurar que sólo un proceso tenga acceso a un recurso compartido en un instante de tiempo.
- Sincronización por condición: Permite bloquear la ejecución de un proceso hasta que se cumpla una condición dada.

33. **a)** ¿Cuál (o cuáles) es el paradigma de interacción entre procesos más adecuado para resolver problemas del tipo "Juego de la vida"?
Respuesta:
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?
Respuesta:
Es conveniente utilizar PMA debido a que con PMS puede ocurrir deadlock y para poder evitarlo debe hacerse mucho mas complicado el algoritmo e ineficiente.

c) ¿Cuál es la arquitectura de hardware que se ajusta mejor? Justifique claramente sus respuestas.
Respuesta:
Se ajusta mejor a SIMD, o sea a arquitecturas de grano fino ya que requiere mucha comunicacion y poco procesamiento.

34. 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)?
Respuesta:
Asignaciones: $128^3 + 128^2 = 2097152 + 16384 = 2113536$
Sumas: $128^3 = 2097152$.
Productos: $128^3 = 2097152$.
Total para $P = 1$ de $ut = 6.291.456$

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.
Respuesta:
6.291.456/8 = 786.432ut asignadas a cada procesador El procesado 8 tarda más 786.432*4 = 3.145.728ut
Speedup = 6.291.456/3.145.728 = 2
Distribuir la carga de acuerdo a los capacidad de cada procesador 128/7 = 18 y 2 de resto
Se asigna 18n a los P1..P7 y 2 a P8

c) Lo mismo que en a) y b) con n=256.
Respuesta:

35. ¿En qué consiste la utilización de relojes lógicos para resolver problemas de sincronización distribuida? Ejemplifique.
Respuesta:
Cada proceso mantiene un reloj lógico local, el cual usa, al recibir un mensaje, para comparar el timestamp del mensaje recibido, comparar la hora y reaccionar en base a eso. 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.

36. **a)** Defina el concepto de sincronización barrier. ¿Cuál es su utilidad?
Respuesta:

Sincronización barrier: una barrera es un punto de demora a la que deben llegar todos los procesos antes de permitirles pasar y continuar su ejecución.

b) ¿Qué es una barrera simétrica?
Respuesta:
Una Barrera Simétrica para n procesos se construye a partir de pares de barreras simples para dos procesos

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, Ej Paridad.

c) Describa combining tree barrier y butterfly barrier. Marque ventajas y desventajas de cada una.
Respuesta:

En un combining tree barrier los procesos se organizan en forma de árbol .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 $2^{(s-1)}$

37. **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.

Respuesta:
La solución de S=P-1 será siempre más estable conforme crezca N por el hecho de que la fórmula de la eficiencia implica S/S_{optimo}, y por definición E debe estar entre 0 y 1, y La resta de P-1 será progresivamente más leve que P/2 conforme crezca P. P/2 implica una eficiencia constante para todo P, pero P-1 será más eficiente.

b) Ahora suponga $S = 1/p$ y $S = 1/p^2$.
Respuesta:
En este caso, 1/P dará un E más grande que $1/P^2$, pues la división sobre un número más grande dará un E menor.

38. 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?
Respuesta:
Ambos programan suman los valores de una matriz, pero cada uno distribuye de manera distinta

la concurrencia del trabajo. Ambos cuentan con un proceso "Acumula" que suma los valores recibidos de "Genera", pero P1 envía todos los valores individuales a "Acumula" para que los sume, mientras que el "Genera" de P2 envía la suma de cada fila.

b) Analice desde el punto de vista del número de mensajes.
Respuesta:
Al enviar todos los valores por separado, P1 envía 10000^2 mensajes mientras que P2, que suma cada fila antes de enviar un mensaje, envía 10000.

c) Analice desde el punto de vista de la granularidad de procesos.
Respuesta:
P1 envía cada valor sin procesarlo, con una gran cantidad de mensajes, por lo que es de grano fino, y P2 suma valores para luego enviar mensajes, por lo que es de grano más grueso.

d) ¿Cuál de los programas le parece el más adecuado para ejecutar sobre una arquitectura tipo cluster PCs? Justifique.
Respuesta:
Los clusters son arquitecturas de grano grueso, con gran caacidad de cómputo, por lo que para aprovechar esto, se debería utilizar el programa P2, que realiza una serie de cálculos antes de enviar un mensaje.

e) Responder las preguntas b, c y d para los siguientes programas.
Respuesta:
No.

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);      }
}
```

39. 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).
Respuesta:
No son equivalentes ya que en el primer segmento puede salir del do si cant esta entre [-10..-1) o (1..10]. El segundo segmento no sale nunca.

b) INCOGNITA equivalente a: (cant > -100)
Respuesta:
Son equivalentes ya que las condiciones de las guardas siempre van a ser verdaderas en ambos segmentos

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

No son equivalentes ya que el valor 0 en el segmento 1 haria que se saldria del do a diferencia del segmento 2 del while if.

d) INCOGNITA equivalente a: ((cant > -10) and (cant < 10))
Respuesta:
No son equivalentes ya que los valores -10 y 10 harian que el segmento 1 termine.

e) INCOGNITA equivalente a: ((cant >= -10) and (cant =< 10))
Respuesta:
Son equivalentes ya que resuelve el problema anterior

40. ¿Qué significa el problema de interferencia en un programa concurrente? ¿Cómo puede evitarse?
Respuesta:
La interferencia es la acción de un proceso que invalida las suposiciones hechas por otro proceso. Puede evitarse sincronizando los procesos, por ejemplo mediante el uso de exclusión mutua entre procesos para acceder estas variables.

41. 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).
Respuesta:

```
1  llegue[N]=([N] false)
2  puedo[N]=([N] false)
3  int i=0
4  process Coordinador{
5      while(true){
6          while(not llegue[i]) i=(i+1) mod N
7          puedo[i]=true
8          while(llegue[i]) {skip}
9      }
10 }
11 process Proceso[id=0..N-1]{
12     --SNC
13     llegue[id]=true
14     while (not puedo[id]) {skip}
15     --SC
16     llegue[id]=false
17     --SNC
18 }
```

42. Clasificación de las arquitecturas multiprocesador:

a) Según el mecanismo de control. Describa.
Respuesta:
SISD (Single Instruction Single Data): Usada por la mayoria de uni procesadores, instrucciones ejecutadas en secuencia siendo la memoria afectada solo por la instruccion del momento, es deterministico

SIMD (Single Instruction Multiple Data): Los procesadores trabajan en paralelo para realizar la misma tarea de manera simultanea y así acelerar el proceso de información.
MISD (Single Instruction Single Data): Ejecutan diferentes instrucciones pero con datos comunes, de forma sincronicas (ej filtros)
MIMD (Multiple Instruction Multiple Data): Los procesadores trabajan en paralela para realizar distintas tareas según sea necesario.

b) Según la organización del espacio de direcciones. Describa.
Respuesta:
Memoria Compartida:
Se relaciona a los metodos de comunicacion de memoria compartida (monitores y semaforos)

- UMA (Uniform Memory Access): Los procesadores acceden a una memoria compartida entre todos los procesos, lo cual unifica el tiempo de acceso para todos los procesos.
- NUMA (Non Uniform Memory Access): Los procesadores se componen de un nodo con su propia memoria local y E/S. El tiempo de acceso a la memoria de otros procesos es distinto del tiempo de acceso a la memoria local.

Memoria Distribuida:
Se relaciona a los metodos de comunicacion de memoria distribuida (pasaje de mensajes), no hay problemas de consistencia, esta asociado a clusters y redes.

c) Según la red de interconexión. Describa.
Respuesta:
Redes estáticas: Constan de links punto a punto y se usan para maquinas de pasaje de mensajes.
Redes dinámicas: Utilizan switches y enlaces de comunicación y se usan para máquinas de memoria de compartidas.

43. 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.
Respuesta:

1) 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.
Respuesta:

```
chan valores[1:n,1:n](int);

Process P[i=1..n;j=1..n]

int v;
int nuevo_min, nuevo_max;
int min = v;
int max = v;
int rondas = 2(n-1);

for(k=1; k <= rondas; k++)
    foreach(vecinos => (x,y))
        send valores[x,y](min,max);
```

```
14
15      foreach(vecinos => (x,y))
16          receive valores[i,j](nuevo_min,nuevo_max);
17          if (nuevo_min < min) min = nuevo_min;
18          if (nuevo_max > max) max = nuevo_max;
19      end;
20  end;
```

2) Analice la solución desde el punto de vista del número de mensajes.
Respuesta:
Las esquinas son 4 (con 2 vecinos cada una).
Los laterales son (N-2) * 4, siendo N el largo de la matriz, y 2 las esquinas de ese lado (con 3 vecinos cada uno), con 4 lados en la matriz.
Los centrales son N*N (toda la matriz), menos los laterales ("N-2" 4 veces), menos las esquinas (4) (con 4 vecinos cada uno).

Cantidad de procesos/mensajes
esquinas= 4 / mensajes+=esquinas*2*rondas
lados= $(n-2)*4$ / mensajes+=lados*3*rondas
centro= $(n^2)-4(n-2)-4$ / mensajes+=centro*4*rondas

3) ¿Puede realizar alguna mejora para reducir el número de mensajes?
Respuesta:
Sí, que los nodos se puedan comunicar con las diagonales. Las esquinas tendrían 3 vecinos, los lados 5, y los del centro 8.

(Preguntas del resumen 2)

44. ¿En qué consiste la propiedad de "A lo sumo una vez" y qué efecto tiene sobre las sentencias de un programa concurrente? De ejemplos de sentencias que cumplan y de sentencias que no cumplan con ASV. Respuesta:

    Si se cumple ASV para una sentencia, ésta se considera atómica.

    En una asignación de tipo x=e
    Donde e puede ser cualquier expresión

```
(1)
x: no es leida por otro proceso
e: tiene como mucho 1 referencia a una variable que es escrita por otro proceso
(2)
x: no es escrita por otro proceso
e: no es escrita por otro proceso
La primera no cumple asv

Cuando no es una asignación, la operación no debe tener más de una variable que es
↪ modificada por otro proceso
Ej: if 1<x+3+y{...} // y=3+2 // x=7+8
```

45. Defina el concepto de "sincronización barrier". ¿Cuál es su utilidad? Mencione alguna de las soluciones posibles usando variables compartidas.
    Respuesta:
    Barrera: punto de sincronización que todos los procesos deben alcanzar para que cualquier proceso pueda continuar. Se utiliza para sincronizar grupos de procesos.

```
int E = log(N);
int arribo[1:N] = ([N] 0);

process P[i=1..N]{
    int j;
    while (true){
        --Sección de código anterior a la barrera.
        --Inicio de la barrera
        for (etapa = 1; etapa <= E; etapa++){
            j = (i-1) XOR (1<<(etapa-1)); --calcula el proceso con cual
            ↪ sincronizar
            while (arribo[i] == 1) → skip;
            arribo[i] = 1;
            while (arribo[j] == 0) → skip;
            arribo[j] = 0;
        }
        --Fin de la barrera
        --Sección de código posterior a la barrera.
    }
}
```

46. Compare la comunicación por mensajes sincrónicos y asincrónicos en cuanto al grado de concurrencia y posibilidad de entrar en deadlock.
Respuesta:
PMA > PMS.
PMA tiene menos chances de entrar en deadlock, por no tener send bloqueante, y mantiene mayor concurrencia por la misma razón. Luego de enviar un mensaje, puede seguir haciendo otras tareas.

47. Defina el concepto de "continuidad conversacional" entre procesos.
Respuesta:
Cuando un proceso necesita hacer consultas a un servidor, una vez establecida la comunicación/-conexión, el cliente realiza todas las consultas necesarias antes de cerrar la conexión para evitar cambios de contextos innecesarios y ahorrar tiempo.

48. Indique por qué puede considerarse que existe una dualidad entre los mecanismos de monitores y pasaje de mensajes. Ejemplifique.
Respuesta:
Se considera que monitores y pasaje de mensajes son equivalentes, ya que uno puede simular el otro, además existen equivalencias entre los mecanismos:

- En monitores hay variables permanentes y en PM variables locales al servidor.
- En monitores se tiene la sentencia wait, la cual encola los procesos que esperan en una variable condición, y en PMA los canales funcionan como colas implícitas de mensajes.
- En monitores existe signal, que despierta a los procesos en espera, y en PMA como PMS, se puede atender el request ya sea con el ? como con el receive de la cola de PMA.

49. Defina el concepto de no determinismo. Ejemplifique.
Respuesta:
El no determinismo es la ausencia de un orden de ejecución preestablecido.

50. ¿Por qué las propiedades de vida dependen de la política de scheduling? ¿Cuándo una política de scheduling es fuertemente fair?
Respuesta:
Las propiedades de vida (ej: terminación) dependen del scheduling, ya que para llegar a un buen estado, las sentencias deben tener la oportunidad de ejecutarse.
Una política de scheduling es fuertemente fair cuando es incondicionalmente fair y las sentencias atómicas condicionales que son verdaderas con infinita frecuencia, en algún momentos se ejecutarán.

51. Defina el problema de sección crítica. Compare los algoritmos para resolver este problema (spin locks, Tie Breaker, Ticket, Bakery). Marque ventajas y desventajas de cada uno.
Respuesta:
La sección crítica existe entre dos o más procesos que intentan acceder a un mismo recurso a la vez, pudiendo causar interferencia. Existen múltiples algoritmos para resolver este problema:

- *Spin Lock:* Su objetivo es hacer atómico el await. Es fácil de implementar pero implica busy waiting, teniendo un procesador dedicado a esperar. Depende de un scheduling fuertemente fair para cumplir eventual entrada.
- *Tie Breaker:* Se suele utilizar con dos procesos. Cada uno chequea cuál es el último en haber intentado entrar y en base a eso logra acceder o no. Utilizar más procesos podría hacer que sea engorroso de escribir y modificar.

- *Ticket:* Es simple para cualquier numero de procesos pero requiere una instruccion especial (fetch and add)
- *Bakery:* Es una variacion de ticket que no utiliza instrucciones especiales pero es mas complejo, aunque mas simple que el tie-breaker de mas de dos procesos.

Tie breaker, ticket y bakery cumplen eventual entrada con un scheduling debilmente fair.

52. Defina procesamiento secuencial, concurrente y paralelo. Ejemplifique en cada caso. Marque especialmente ventajas e inconvenientes en cada uno de ellos.

Procesamiento secuencial: Hay un orden temporal estricto, las instrucciones se ejecutan una después de la otra
Procesamiento concurrente: No hay orden estricto, el programa se dividen en procesos que se ejecutan al mismo tiempo, compitiendo por el procesador
Procesamiento paralelo: No hay orden estricto, el programa se divide en varios procesos que son asignados a varios procesadores.

53. Defina scheduling, deadlock, fairness, inanición y overloading. Ejemplifique cada uno.
Respuesta:

- *Scheduling:* Asignacion de recursos a diferentes procesos segun alguna politica (ej: SJF, prioridad, etc)
- *Deadlock:* Dos o mas procesos que estan esperando por un recurso compartido. ej: proceso 1 tiene un recurso que precisa el proceso 2 y para dejar de usarlo precisa un recurso que tiene el proceso 1. Ambos procesos estan esperando algo que nunca va a ocurrir.
- *Inanicion:* Se da cuando la politica de scheduling no le asigna nunca recursos a cierto proceso. ej: Politica SJF, a un proceso que tarda mucho puede que nunca le asignen recursos si procesos que tardan poco siempre llegan antes que este se pueda ejecutar.
- *Overloading:* Se da cuando la politica de scheduling le asigna una carga que excede la capacidad de procesamiento del recurso. ej: politica que le asigna quantums chicos a un procesador y grandes a otro, vienen muchos procesos con quantums chicos por lo que el primer procesador esta overloaded.

54. Analice qué tipo de mecanismo de pasaje de mensajes son más adecuados para resolver problemas de tipo cliente-servidor, pares que interactúan, filtros y productores-consumidores. Justifique
Respuesta:

- *Cliente-servidor:* Es mejor con PMA ya que en PMS este puede tener una demora innecesaria cuando el cliente este liberando el recurso y tampoco podria hacer cosas entre medio del envio de mensajes.
- *Pares que interactuan:* Es mejor con PMA ya que al ser que los pares hacen todos lo mismo, uno puede terminar antes que otro. Con PMS podria haber demora innecesaria entre uno que envíe y otro que reciba.
- *Filtros:* Es mejor con PMA ya que cuando un proceso termina su trabajo puede haber demora en la comunicación entre el que realizo la tarea y el siguiente filtro haciendo que pueda perderse tiempo a la hora de realizar nuevas tareas.
- *Productores Consumidores:* Es mejor con PMA ya que la velocidad del productor y la del consumidor pueden ser distintas. Con PMS uno tendría que esperar al otro para seguir haciendo sus tareas, u usar un buffer intermedio.

55. Describa el paradigma "bag of tasks". ¿Cuál es la principal ventaja del mismo?
Respuesta:
Bag of tasks consiste en una bolsa de tareas de la cual cada proceso puede tomar una tarea cuando este libre y realizarla. La principal ventaja es reducir el tiempo de ejecución ya que los procesos que terminen mas rápido sus tareas pueden realizar otras.

56. Defina monitores. Diferencie monitores y semáforos.
Respuesta:
Monitores: es una herramienta que permite abstraer el acceso a un recurso mediante una interfaz definida (procedimientos/modulos). Tiene exclusión mutua de forma implícita cuando los procesos intentan acceder al monitor.

Diferencias:

- Monitores tiene la ventaja de ser más abstracto y fácil de utilizar ya que se lo puede ver al monitor como una caja negra sin ver realmente lo que pasa adentro, pudiendo agregar procesos sin preocuparme de que las variables compartidas se modifiquen de forma incorrecta.
- Evita la interferencia de variables compartidas a través de la exclusión mutua implícita cosa que en semáforos es tarea del programador realizarlo, lo que puede llevar a errores humanos.
- En monitores existen variables condición que permiten dormir al proceso dentro del monitor. Estas tienen una cola de orden FIFO que permite registrar el orden de llegada de manera implícita.

57. ¿Qué relación encuentra entre el paralelismo recursivo y la estrategia de dividir y conquistar? ¿Cómo aplicaría este concepto a un problema de ordenación de un arreglo?
Respuesta:
El *paralelismo recursivo* es un paradigma muy apto para la estrategia de dividir y conquistar, ya que consiste en asignar un proceso nuevo a cada llamada recursiva, dividiendo el problema naturalmente en los procesos. Esto se podría adaptar a la ordenación de un arreglo de la siguiente manera:
Fase de dividir:
Se divide el arreglo en dos mitades y se asigna una a cada uno de 2 procesos nuevos. Esto se repite hasta que solo queda 1 numero en cada proceso.
Fase de conquistar:
Los procesos devuelven a su padre el array ordenado, y este a su vez ordena los resultados de sus hijos hasta llegar al proceso original.

58. ¿Qué condiciones deben darse para que ocurra deadlock?
Respuesta:
Deben darse 3 de las siguiente 4 situaciones:

- *Recursos reusables serialmente:* Los procesos comparten mas de un recurso que pueden usar con exclusión mutua.
- *Adquisición incremental:* El proceso adquiere sus recursos de a uno a la vez.
- *No-preemption:* mantienen recursos obtenidos y no se los pueden quitar otros de manera forzada.
- *Espera cíclica:* cadena circular donde un proceso tiene un recurso que el sucesor precisa indefinida cantidad de veces durante todo el ciclo.

(Preguntas del examen de marzo 2022)

59. Conceptos generales

a) Escribir una definicion de programación concurrente y programación paralela, diferenciando ambos conceptos.
Respuesta:
La programación concurrente implica crear dos o más programas secuenciales que pueden o no ejecutarse en paralelo, que interactúan intercambiando información para cumplir un objetivo en común.
La programación paralela implica ejecutar estos programas efectivamente en distintos procesadores simultáneamente para reducir el tiempo de ejecución total del programa.

b) Definir lo que es la sincronización entre procesos y explicar cuales son los dos tipos de sincronización en que se clasifica. Dar un ejemplo (explicarlo con palabras, no código) de un problema donde se usen ambos tipos de sincronización.
Respuesta:
La sincronización es la posesión de información acerca de otro proceso para coordinar actividades.
Existen dos tipos:
· Por condición: Permite restringir las historias de un programa concurrente para asegurar el orden temporal necesario.
Ej: Antes de comenzar, que un proceso chequee que todos los demás procesos hayan llegado al mismo punto de ejecución.
· Por exclusión mutua: Permite ejecutar sentencias como una única acción atómica, evitando de esta manera la interferencia entre procesos.
Ej: Si se tiene un recurso que puede usarse por un solo proceso a la vez, se utilizan herramientas que garanticen atomicidad de las acciones de ese proceso sobre el recurso.

c) Definir comunicación entre procesos. Explicar cuáles son los dos tipos de comunicación que existen.
Respuesta:
La comunicación entre procesos es el envío y la recepción de información para facilitar la sincronización entre procesos.
Los tipos de comunicación son:
· Memoria compartida: Los procesos comparten un espacio de memoria y se comunican por variables globales.
· Pasaje de mensajes: Los procesos envían mensajes a través de redes de comunicación, generalmente se usa en arquitecturas distribuidas.

d) Indicar que es la programación distribuida y cuáles son las características que lo diferencian de la programación con memoria compartida.
Respuesta:
La programación distribuida es aquella en la que los procesos que ejecutan un programa concurrente se comunican por mensajes. Supone la ejecución sobre una arquitectura de memoria distribuida, aunque puedan ejecutarse sobre una de memoria compartida (o híbrida).
A diferencia de la programación con memoria compartida, en la distribuida se utilizan mensajes, en vez de variables compartidas, para la sincronización y comunicación, y debido a la transmisión de datos a procesos alocados externamente, es más lenta que la programación con

memoria compartida.

60. Problema de la sección crítica

a) Nombrar y explicar las 4 propiedades que debe cumplir una solución a este problema. Para cada una de estas 4 propiedades explicar la idea de una solución que no cumpla dicha propiedad (con palabras, no con código).
Respuesta:

- **Exclusión mutua** A lo sumo un proceso está en su SC.
  Una solución que no cumple: En semáforos, varios procesos modifican el valor de una variable sin garantizar previamente la atomicidad de sus acciones por medio de un semáforo.
- **Ausencia de deadlock** si 2 o más procesos tratan de entrar a sus SC, al menos uno tendrá éxito.
  Una solución que no cumple: Varios procesos realizan incrementalmente la adquisición de los recursos que necesitan utilizar, pudiendo haber varios que requieran uno que otro ya tomó primero, y que no liberará hasta tener uno que ese proceso en espera tiene.

- **No demora innecesaria** si un proceso trata de entrar a su SC y los otros no, el primero no está impedido de hacerlo.
  Una solución que no cumple: Los procesos son coordinados por un timer global que mide X tiempo (por ej. 10 segs) antes de que se pueda dejar entrar a un proceso a la SC.

- **Eventual entrada** Un proceso que intenta entrar a su SC tiene posibilidades de hacerlo.
  Una solución que no cumple: Un programa se maneja por medio de un scheduler SJN sin ageing, dando lugar a que infinitos procesos breves lleguen antes que uno muy largo.

b) Desarrollar una solución de GRANO FINO usando sólo variables compartidas (no usar await, sentencias especiales como TS, semáforos o monitores) usando un coordinador. En este caso, cuando un proceso SC[i] quiere entrar a su sección critica le avisa al coordinador, y espera a que éste le dé permiso. Al terminar de ejecutar su sección critica, el proceso SC[i] le avisa al coordinador.
Respuesta:

```
1  llegue[N]=([N] false)
2  puedo[N]=([N] false)
3  int i=0
4  process Coordinador{
5      while(true){
6          while(not llegue[i]) i=(i+1) mod N
7          puedo[i]=true
8          while(llegue[i]) {skip}
9      }
10 }
11 process Proceso[id=0..N-1]{
12     --SNC
13     llegue[id]=true
```

```
14      while (not puedo[id]) {skip}
15      --SC
16      llegue[id]=false
17      --SNC
18  }
```

61. Monitores: indicar como se realiza en monitores la comunicación y la sincronización (ambos tipos de sincronización) entre procesos. Explicar la diferencia entre los protocolos de sincronización en monitores: signal and wait (S&W) y signal and continue (S&C).
Respuesta:
En monitores, la comunicación se realiza a través del llamado a procedures del mismo por parte de los procesos. Solo un proceso puede acceder a la vez al monitor, lo cual nos da una exclusión mutua implícita en la herramienta. Una vez dentro, si las circunstancias no son las necesarias, el proceso puede dormirse en una variable condición, a la espera de ser llamado (signal).
En signal and wait, el proceso llamador deja el procesador para darle espacio al proceso llamado, y el primero pasa a competir por el procesador con el resto.
En signal and continue, el proceso llamador, luego de despertar a otro y que este pase a competir con el resto, permanece en el procesador hasta finalizar su tarea, y sale del monitor.

62. PMA y PMS: Explicar el funcionamiento general de PMA y PMS. Indicar y explicar las sentencias que posee para realizar las comunicaciones básicas.
Respuesta:

PMA: Pasaje de mensajes asincronicos: Es un estilo de pasaje de mensajes con canales de tipo mailbox donde hay varios emisores y varios receptores, con una cola implícita de mensajes no atendidos. Posee las primitivas send() que es un envío no bloqueante osea no debe esperar a que se reciba el mensaje y receive() que es bloqueante y debe recibir un mensaje para continuar con la ejecución.
PMS: Pasaje de mensajes sincrónicos: A diferencia de PMA, PMS tiene canales punto a punto de tipo link donde solo hay un emisor y un receptor, con un canal implícito por cada par de procesos. Posee las primitivas ! para enviar mensajes, este es un envío bloqueante y debe ser recibido antes de poder seguir; para recibir se usa ? que también es bloqueante.

63. RPC y Rendezvous: Explicar las características comunes y las diferencias entre Rendczvous y RPC. Indicar que cosas de la comunicación guardada de rendezvous "conceptual" no se tienen en el rendezvous provisto, por ADA.
Respuesta:
· En común: Ambos tienen canales bidireccionales implícitos, trabajan con memoria distribuida.
· Diferencias: Rendezvous cuenta con un único proceso servidor que atiende los pedidos, mientras que en RPC, el proceso servidor se encargar de generar nuevos procesos hijos que los atiendan. Además, RPC es únicamente un mecanismo de comunicación, por lo tanto requiere implementar un sistema de sincronización aparte (como puede ser semáforos).

64. Librería Pthreads: explicar cómo se maneja la sincronización (ambos tipos de sincronización) en la librería Pthreads, como se relacionan entre ellas y por qué para realizar una de las sincronizaciones necesita de la otra?.
Respuesta:
POSIX o Pthreads cuenta con la sincronización por exclusión mutua, a modo de semáforos -con la instrucción pthread_mutex_lock(&mutex) y pthread_mutex_unlock(&mutex)- y la sincronización

por condición -pthread_cond_wait(pthread_cond_t cond, pthread_mutex_t *mutex) y pthread_-cond_signal (pthread_cond_t cond)-, la cual requiere (y se puede ver en sus parámetros) del otro tipo de sincronización, un mutex. Esto se debe a que no existen monitores como se han visto en la materia, sino que estos se emulan mediante el uso de semáforos y extracción de métodos.

65. Librerías para Pasaje de Mensajes: Explicar las 4 diferentes tipos de SEND en que se clasifican las comunicaciones punto a punto en las lihrerias de Pasaje de Mensajes en general (no se refiere particularmente a MPI). Indicar cuales se corresponden con las herramientas PMA y PMS de la práctica 4.
Respuesta:
En las librerías de PM existen:
· Send bloqueante con buffering.
· Send bloqueante sin buffering.
· Send no bloqueante con buffering.
· Send no bloqueante sin buffering.
En la práctica 4 de la materia usamos send bloqueante sin buffering (PMS) y send bloqueante con buffering (PMA).

66. Paradigmas de interacción: Suponga que una imagen se encuentra representada por una matriz A(nxn), y que el valor de cada pixel es un número entero que es mantenido por un proceso distinto (es decir, el valor del pixel (i,j) esta en el proceso P(i,j)). Cada proceso puede comunicarse a lo sumo con sus 8 vecinos que lo rodean (o la cantidad que tenga si está en los bordes). Escriba un algoritmo heartbeat que calcule el máximo y el mínimo valor de los pixels de la imagen. Al terminar el programa. cada proceso debe conoce ambos valores.
Respuesta:

```
chan valores[1:n,1:n](int);

Process P[i=1..n;j=1..n]

int nuevo, k;
int vecinos[n][n] = conocerVecinos();
int v = conocerValor();
int min = v;
int max = v;
int rondas = n-1;

for(k=1; k <= rondas; k++)
    foreach((x,y) in vecinos)
        send valores[x,y](min, max);
    foreach(vecinos)
        receive valores[i,j](nuevoMin, nuevoMax);
        if (nuevo < min)
            min = nuevo;
        if (nuevo > max)
            max = nuevo;
        end;
    end;
```

```
end;
```

67. Metricas en sistemas paralelos: Sea la siguiente solucion al problema del producto de matrices de nxn con P procesos en paralelo con variables compartidas. Suponga n=640 y cada procesador capaz de ejecutar un proceso.

```
process worker[w=1 to P]{
    int primera = (w-1)*(n/P)+1
    int ultima = primera+(n/P)-1

    for [i = primera to última]{
        for [j=1 to n]{
            c[i,j]=0
            for [Z=1 to n]{
                c[i,j]=c[i,j]+(a[i,Z]*b[Z,j])
            }
        }
    }
}
```

a) Calcular cuántas asignaciones, sumas y productos se hacen secuencialmente (caso en que P=1), y cuántas se realizan en cada procesador en la solución paralela con P=8.
Respuesta:
En caso de que sea $P = 1$
Asignaciones: n^3
Sumas: n^3
Multiplicaciones: n^3
Total: $3n^3$
En caso de que sea $P = 8$
Asignaciones: $n^3/8$
Sumas: $n^3/8$
Multiplicaciones: $n^3/8$
Total: $3n^3/8$

b) Dado que los procesadores de PI y P2 son idénticos, con tiempos de asignación 2, de suma 4 y de producto 8; los procesadores P3, P4, P5 y P6 son la mitad de potentes (tiempos 4, 8 y 16 para asignaciones, sumas y productos respectivamente); y los procesadores P7 y P8 son el doble de potentes que P1 (tiempos 1, 2 y 4 para asignaciones, sumas y productos respectivamente). Calcular cuánto tarda el programa paralelo y el secuencial
Respuesta:
Solución Secuencial: Se usa P7 ->
Asignaciones: $n^3 * 1$
Sumas: $n^3 * 2$
Multiplicaciones:$n^3 * 4$
Total:$n^3 + 2n^3 + 4n^3 = 7n^3 ut = 7 * 640^3 = 1835008000 ut$
Solución Paralela:
Tiempo P1..2: $2n^3/8 + 4n^3/8 + 8n^3/8 = 14/8n^3$
Tiempo P3..6: $4n^3/8 + 8n^3/8 + 16n^3/8 = 28/8n^3$

Tiempo P7..8: $n^3/8 + 2n^3/8 + 4n^3/8 = 7/8n^3$
Tiempo paralelo: $max(TiempoP1..2, TiempoP3..6, TiempoP7..8) = 28/8n^3 = 7/2n^3$

c) Realizar paso a paso el cálculo del valor del speedup y la eficiencia.
Respuesta:
Speedup: $T_{seq}/T_{par} = 7n^3/(7/2n^3) = 7 * 7/2 = 2$
$S_{optimo} = 2 * [(7/8n^3)/(14/8n^3)] + 4 * [(7/8n^3)/(28/8n^3)] + 2 * [(7/8n^3)/(7/8n^3)] = 1 + 1 + 2$
Eficiencia: $Speedup/S_{optimo} = 2/4 = 50\%$

d) Modificar el código para lograr una mejor eficiencia. Calcular la nueva eficiencia y comparar con la calculada en (c).
Respuesta:
$4 * x + 2 * 2x + 2 * 4x = 1$
$4x + 4x + 8x = 1$
$x = 1/16$ siendo x la unidad mínima de trabajo

Carga P1..2: 2*1/16 = 1/8
Carga P3..6: 1/16
Carga P7..8: 4*1/16 = 1/4

Tiempo P1..2: $2n^3/8 + 4n^3/8 + 8n^3/8 = 14/8n^3$
Tiempo P3..6: $4n^3/16 + 8n^3/16 + 16n^3/16 = 28/16n^3$
Tiempo P7..8: $n^3/4 + 2n^3/4 + 4n^3/4 = 7/4n^3$
Tiempo paralelo: Cualquiera, son iguales

Speedup: $T_{seq}/T_{par} = 7n^3/(7/4n^3) = 7 * 7/4 = 4$
Eficiencia: $Speedup/S_{optimo} = 4/4 = 100\%$

Modificacion del codigo

```
-- Factor es un array que conoce el factor de
-- carga calculado arriba para cada procesador
process worker[w=1 to P]{
    int primera = (w-1)*n*factor[w]+1
    int ultima = primera+n*factor[w]-1

    for [i = primera to última]{
        for [j=1 to n]{
            c[i,j]=0
            for [Z=1 to n]{
                c[i,j]=c[i,j]+(a[i,Z]*b[Z,j])
            }
        }
    }
}
```

NOTA: para hacer los cálculos sólo tenga en cuenta las operaciones realizadas en la instrucción dentro del for Z

(Final de Programación Concurrente de diciembre:)

68. Propiedad a A Lo Sumo una Vez (ASV)
    Respuesta:
    La propiedad de a lo sumo una vez indica que para una sentencia de asignación, solo se puede tener una referencia crítica que puede ser referenciada a lo sumo 1 vez. Ej:
    x=y+1//y=2//z=y+3
    Esto cumple x=y+1//y=2//z=y+x
    Esto no cumple

69. Alocación de recursos. Es SJF fair? Cómo se resuelve?
    Respuesta:
    SJF No es fair, porque podrían presentarse infinitos procesos de alta prioridad frente a uno de baja prioridad que, en consecuencia, nunca se ejecutaría.
    Se resuelve agregando ageing. Que consiste en agregar una prioridad extra para los procesos que estuvieron por más tiempo en el procesador.

70. Qué son RPC y Rendevouz y para qué se usan.
    Respuesta:
    Son mecanismos de comunicación -y en el caso de Rendezvous, también sincronización-, que se usan para paradigmas como Cliente-Servidor en los que la interacción suele requerir una respuesta inmediata.

71. Comunicación guardada.
    Respuesta:
    En PM, cuando se necesita decidir si es correcto recibir un mensaje, se pueden utilizar condiciones booleanas que protegen la recepción del mensaje, que es bloqueante. Ambas cosas en combinación forman la guarda, cuya función es garantizar la posible recepción de un mensaje. Si la condición booleana es true, pero no hay mensajes para recibir, la guarda permanece bloqueada hasta recibir algo.
    La comunicación guardada de CSP se basa en guardas de la forma B; (C) S donde B es la condición booleana, que se asume true si está ausente, C es una sentencia de comunicación y S son las sentencias que se ejecutan cuando la guarda es exitosa, la guarda además puede fallar si la condición booleana es falsa y puede ser bloqueada si la sentencia de comunicación se demora.

72. Definición de speedup y eficiencia y además calcular uno si el máximo pararelizable es algo como el 80%.
    Respuesta:
    El speedup es la relación entre los tiempos de ejecución de un algoritmo ejecutado en forma secuencial y la solución paralela al mismo problema.
    La eficiencia mide la fraccion de tiempo en que los procesadore son utiles para el computo
    Si la cantidad de maximo pararelizable es 80% con la ley de Amdahl se puede sacar el limite que es:

$$1/(1 - 0.8) = 5$$

73. IMPORTANTE: este ejercicio decía el enunciado que lo tenías que tener bien sí o sí para aprobar, "sin importar" que el resto del final esté bien:
    Ejercicio de matrices NxN con P procesadores:

```
process worker[w=1 to P]{
    int primera = (w-1)*(n/P)+1
    int ultima = primera+(n/P)-1

    for [i = primera to última]{
        for [j=1 to n]{
            c[i,j]=0
            for [Z=1 to n]{
                c[i,j]=c[i,j]+(a[i,Z]*b[Z,j])
            }
        }
    }
}
```

a) Cantidad de asignaciones, sumas y productos con P1 y P2 con algunos procesadores más lentos y otros más rápidos.
Respuesta:

Formula general (basado en el código): $n/P * n * n$
Con $P = 1$

$$asig = n^3$$

$$sum = n^3$$

$$mult = n^3$$

Con $P = 8$

$$asig = n^2 * n/8$$

$$sum = n^2 * n/8$$

$$mult = n^2 * n/8$$

b) Tiempo de ejecución con P1 a P6 con operaciones 1,2,3 y P7 y P8 toman el triple de tiempo.
Respuesta:

Formula general (basado en el código):
$asignaciones * T_{asig} + sumas * T_{suma} + multiplicaciones * T_{mult}$
$S_{optimo} = \sum_{i=0}^{P} \frac{PotenciaCalculo(mejor)}{PotenciaCalculo(i)}$
Tiempo de P1-P6

$$6n^2 * n/8$$

Tiempo de P7-P8

$$18n^2 * n/8$$

Tiempo ejecucion de forma paralela: $max(18n^2 * n/8, 6n^2 * n/8) = 18n^2 * n/8$

c) Calcular speedup. Calcular eficiencia
Respuesta:
Fórmula $Speedup = T_{seq}/T_{par}$

$$speedup = 6n^3/(18n^2 * n/8)$$
$$speedup = n^3/3n^2 * n/8$$
$$speedup = n/3 * n/8$$
$$speedup = 8n/3n$$
$$speedup = 8/3 = 2,\widehat{6}$$
$$S_{optimo} = 2 * [(6n^2 * n/8)/(18n^2 * n/8)] + 6 * [(6n^2 * n/8)/(6n^2 * n/8)] = 2/3 + 6 = 6,\widehat{6}$$
$$Eficiencia = \frac{S}{S_{optimo}} = 2,\widehat{6}/6,\widehat{6} = 0,4 = 40\%$$

d) Cómo se podría mejorar el speedup?
Respuesta:

P7-P8 con la carga minima (x)
P1-P6 con 3 veces mas carga $(3x)$
Balanceando la carga: $2 * x + 6 * 3x = 1$ $(1 = 100\%)$
$20x = 1$
$x = 1/20 \rightarrow$ Unidad mínima de trabajo
P7-P8 con carga $1/20$
P1-P6 con carga $3/20$
Tiempo de P1-P6
$6n^2 * 3n/20$
Tiempo de P7-P8
$18n^2 * n/20$

$$speedup_{new} = 6n^3/6n^2 * 3n/20$$
$$speedup_{new} = n/3n/20$$
$$speedup_{new} = 20n/3n$$
$$speedup_{new} = 20/3 = 6,\widehat{6}$$
$$Eficiencia = \frac{S}{S_{optimo}} = 6,\widehat{6}/6,\widehat{6} = 100\%$$