1) Defina programa: Concurrente, Paralelo, 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.
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) Defina comunicación entre procesos. Describa los mecanismos que conozca.

La comunicación entre procesos concurrentes indica el modo en que se organiza y trasmiten datos entre tareas concurrentes. Los procesos se comunican por:

• **Memoria Compartida:** Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos que hay en ella. Logicamente 2 procesos no pueden operar al mismo tiempo sobre ella.

• **Pasaje de Mensajes:** Es necesario tener un canal físico o lógico por el cual los procesos se enviarán mensajes, así como también un protocolo de comunicación.

3) Defina sincronización entre procesos. Describa los mecanismos que conozca.

La sincronización es la posesión de información acerca de otro proceso para coordinar actividades. Los procesos se sincronizan por:

• **Por exclusión mutua**: Asegurar que sólo un proceso puede estar en la sección crítica en un instante de tiempo.

• **Por condición:** Bloquea la ejecución de un proceso hasta que se cumpla una condición dada.

4) Que significa el problema de la interferencia? Cómo puede evitarse?

Es cuando un proceso realiza una acción que invalida las suposiciones que habia hecho otro proceso. Por ejemplo en programas concurrentes con memoria compartida.
Para evitarse, se debe asegurar un orden temporal entre las acciones que ejecutan los procesos. A esto se le llama sincronización, y existen 2 mecanimos: *exclusión mutua* y *por condición.*

5) En que consiste la propiedad de "a lo sumo una vez"? En que afecta a un programa concurrente? Ejemplos que cumplan y que no cumplan ASV.

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

int x=0, y=0; co **x=x+1** // **y=y+1** oc;	No hay ref. críticas en ningún proceso. En todas las historias x = 1 e y = 1
int x = 0, y = 0; co **x=y+1** // **y=y+1** oc;	El 1er proceso tiene 1 ref. crítica. El 2do ninguna Siempre y = 1 y x = 1 o 2
int x = 0, y = 0; co **x=y+1** // **y=x+1** oc;	Ninguna asignación satisface *ASV*. Posibles resultados: x= 1 e y =2 / x = 2 e y = 1 **Nunca debería ocurrir x = 1 e y = 1 → ERROR**

Si no satisface esta propiedad entonces deber ejecutarse **atómicamente.**

6) Por qué las propiedades de vida dependen de las políticas de scheduling? Cuándo un política de scheduling es fuertemente fair?

Una propiedad de un programa concurrente es un atributo verdadero en cualquiera de las historias de ejecución del mismo. Toda propiedad puede ser formulada en términos de dos clases:

- **seguridad (safety)**
  - Nada malo le ocurre a un proceso: asegura estados consistentes.
  - Ejemplos: ausencia de deadlock y ausencia de interferencia (exclusión mutua) entre procesos, partial correctness.
- **vida (liveness)**
  - Eventualmente ocurre algo bueno con una actividad: progresa, no hay deadlocks.
  - Una falla de vida indica que las cosas dejan de ejecutar.
  - Ejemplos de vida: terminación, asegurar que un pedido de servicio será atendido, que un mensaje llega a destino, que un proceso eventualmente alcanzará su SC, etc ⇒ **dependen de las políticas de scheduling.** Porque son ellas las que determinan cuál próxima instrucción atómica va a ejecutarse.

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

Fairness Incondicional: Una política de scheduling es incondicionalmente fair si toda acción atómica **incondicional** que es elegible, eventualmente es ejecutada.

Fairness Débil: Una política de scheduling es débilmente fair si :

1) **Es incondicionalmente fair y**
2) Toda acción atómica **condicional** que se vuelve elegible eventualmente es ejecutada, asumiendo que su condición se vuelve **true** y **permanece true** hasta que es vista por el proceso que ejecuta la acción atómica condicional.

*No es suficiente para asegurar que cualquier sentencia **await elegible eventualmente** se ejecuta: **la guarda podría cambiar** el valor (de false a true y nuevamente a false) **mientras un proceso está demorado**.*

Fairness Fuerte: Una política de scheduling es fuertemente fair si:

1) Es **incondicionalmente fair** y
2) Toda acción atómica **condicional** que se vuelve elegible eventualmente es ejecutada pues su guarda se convierte en true con **infinita frecuencia.**

7) Describa la técnica de passing the condition. Ejemplifique.

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) pasandole una condición.

8) En que consiste la técnica barrier? Mencione al menos 2 soluciones posibles usando variables compartidas.

Son un punto de encuentro (sincronización) en donde todos los procesos se demoran hasta que todos hayan llegado y luego de eso puedan continuar su ejecución.

9) Que se entiende por aquitectura de grano grueso? Es más adecuado para programas con mucha o poca comunicación?

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 **poca comunicación.**

10) Qué significa que un problema sea de exclusión mutua selectiva? El problema de los filósofos tradicional, lo es? Y si hubiera 3?

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.

11) En que consiste la comunicación guardada y cual es su utilidad? Describa como son las sentencias alternativas e iterativas con comunicación guardada.

Con frecuencia un proceso se quiere comunicar con más de uno de otros procesos (quizás por distintos ports) y no sabe el orden en el cual los otros procesos podrían querer comunicarse con él. 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.

Las sentencias de comunicación guardadas aparecen en **if** y **do**.

if B1; comunicación1 → S1;
 B2; comunicación2 → S2;
fi

Ejecución:

 Primero, se evalúan las guardas.
1) Si todas las guardas fallan, el if termina sin efecto.
2) Si al menos una guarda tiene éxito, se elige una de ellas (no determinísticamente).
3) Si algunas guardas se bloquean, se espera hasta que alguna de ellas tenga éxito.
 Segundo, luego de elegir una guarda exitosa, se ejecuta la sentencia de comunicación de la guarda elegida.
 Tercero, se ejecuta la sentencia S i .

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

13) Práctico

14) Práctico

15) Práctico

16) Defina el problema de la sección crítica. Compare los algoritmos para resolver estos problemas (spin locks, tie breaker, ticket, bakery)

En este problema, n procesos repetidamente ejecutan una sección crítica de código, luego una sección no crítica. La sección crítica está precedida por un protocolo de entrada y seguido de un protocolo de salida.
Cada sección crítica es una secuencia de sentencias que acceden algún objeto compartido. Cada sección no crítica es otra secuencia de sentencias. Asumimos que un proceso que entra en su sección crítica eventualmente sale; así, un proceso solo puede finalizar fuera de su sección crítica.

Spin locks: los procesos se quedan iterando (spinning) mientras esperan que se limpie lock.
- Baja performance en multiprocesadores si varios procesos compiten por el acceso.
- lock es una variable compartida y su acceso continuo es muy costoso (“memory contention”).
- Además, podría producirse un alto overhead por cache inválida
- Fácil de implementar

Tie breaker: Requiere solo scheduling incondicionalmente fair para satisfacer la propiedad de eventual entrada. Además no requiere instrucciones especiales del tipo Test-and-Set. Sin embargo, el algoritmo es mucho más complejo que la solución spin lock.

Ticket: Se basa en repartir tickets (números) y luego esperar turno. Cuenta con el problema de que el contador de ticket crezca hasta un overflow (puede resolverse reseteando el contador, una vez alcanzado el número). Es más facil de implementar que el completo Tie breaker.

Bakery: Algoritmo del tipo de ticket, fair y no requiere instrucciones de máquina especiales. El algoritmo es más complejo que el ticket, pero ilustra una manera de romper empates cuando dos procesos obtienen el mismo número.

17)

a) Describa brevemente para que sirven los algoritmos de RPC y Rendevouz. Para que tipos de problemas son mas 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 c*omunicación como de sincronización.*
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 de mecanismos de sincronización en módulos de RPC? Cómo puede realizarse esta sincronización?

Es necesario proveer sincronización dentro de los módulos porque los procesos pueden ejecutar concurrentemente. Existen dos formas de sincronizar estos procesos:
Si los procesos se ejecutan por exclusión mutua, sólo 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 elementos de la forma general de rendezvous no se encuentran en el lenguaje ADA?

Ada no 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.

18) Explique sinteticamente los 7 paradigmas de interacción entre procesos en la programación distribuida. En cada caso, ejemplifique, indique que tipo de comunicación por mensajes es mas conveniente. Justificar.

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.

19)

a) Defina las metricas de speedup y eficiencia. Que miden? Cual es el rango de valores de 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 cuanto más rápido es el algoritmo paralelo con respecto al algoritmo secuencial, es decir cuánto se gana por usar más procesadores. Cuando más se acerque a p, mejor es la solucion paralela.

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.

¿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.

b) Que se entiende por escalabilidad de un sistema paralelo?

Da una medida de usar eficientemente un número creciente de procesadores.

c) Suponga que se tiene dos soluciones paralelizables por p procesadores S=p-4 y la otra S=p/2. Cual de las 2 se comportará mas eficientemente cuando incremente el numero de procesadores?

Por definicion de speedup (cuanto mas cercano a p, mas eficiente) es la solucion **S=p-4**. Si comparamos las eficiencias tenemos que: E=S/p, entonces E=p-4/p y E=p/2/p, cuando mas grande sea p, mayor eficiencia tendrá el primero, ya ue la primera sera cercano a 1 y la otra no pasará de la mitad.

d) Suponga que el tiempo de ejecución de un algoritmo secuencial es de 1000 unidades de tiempo, de las cuales sólo el 90% es paralelizable. Cuál es el limite en la mejora que se puede obtener paralelizando el algoritmo? Justifique.

El limite de mejora se da teniendo 900 procesadores los cuales ejecutan una unidad de tiempo obteniendo un tiempo paralelo de 101 unidades de tiempo. El speedup mide la mejora del tiempo obtenida con un algoritmo paralelo frente al secuencial. **S = Sc / Sp:** S = 1000 / 101 = 9,90 ~ 10, aunque se utilicen mas procesadores el mayor speedup alcanzable es el anterior cumpliendose asi la ley de Amdahl que dice que el limite esta por la cantidad de codigo secuencial y no por la cantidad de procesadores.

REVISAR

20) Practico