Índice

TEMA 1: Introducción a la programación concurrente y paralela

1.¿Qué es la programación concurrente?

Explica qué es un hilo (thread) detallando sus propiedades

Un hilo es un proceso ligero que se ejecuta en el mismo espacio de memoria que otros hilos. Tienen una pila de ejecución propia y consumen menos recursos que los procesos pesados.
A su vez, son menos robustos ya que al compartir el espacio de direcciones y el contexto de ejecución entre hilos, un fallo en uno afecta al resto.
También consumen menos recursos y tardan menos en inicializarse que los procesos pesados ya que comparten sus recursos con otros hilos y con el padre. La comunicación entre hilos también es más fácil y consume menos recursos por esta razón.

Pregunta 1.- Señale cuál de las siguientes afirmaciones es FALSA, respecto a los procesos:

a) No dispone de pila de ejecución.
b) Cada proceso tiene su propio espacio de memoria.
c) Están gestionados por el sistema operativo.
d) Son unidades autónomas e independientes.

Pregunta 3.- Señale cuál de las siguientes afirmaciones es VERDADERA:

a) Los hilos consumen más recursos que los procesos.
b) Los procesos tardan más tiempo en inicializarse que los hilos.
c) Los hilos son más robustos que los procesos.
d) Ninguna de las respuestas anteriores es correcta.

Pregunta 1.- Señale cuál de las siguientes afirmaciones es FALSA, respecto a la programación concurrente:

a) Nos permite aprovechar el procesador cuando un proceso espera por una operación de entrada/salida.
b) Resuelve los problemas de exclusión mutua y comunicación entre procesos.
c) Permite ofrecer un entorno interactivo a múltiples usuarios.
d) Los lenguajes de programación y sus librerías evolucionaron para poder desarrollar programas concurrentes.

Pregunta 3.- Seleccione la opción VERDADERA:

a) Un proceso es la ejecución de un hilo en un sistema informático
b) Cada hilo tiene su propio espacio de memoria
c) Cada hilo tiene su propia pila de ejecución
d) Todas las anteriores son falsas

Pregunta 2.- Seleccione la opción VERDADERA respecto a la concurrencia

a) Se dice que dos procesos son concurrentes si la primera instrucción de uno de ellos se ejecuta antes de la primera y la última instrucción del otro.
b) Se dice que dos procesos son concurrentes si la primera instrucción de uno de ellos se ejecuta después de la primera instrucción del otro.
c) Se dice que dos procesos son concurrentes si la última instrucción de uno de ellos se ejecuta después de la última instrucción del otro.
d) Ninguna de las anteriores es VERDADERA

Pregunta 3.- Escoja la respuesta FALSA respecto a los hilos:

a) Todos los hilos de un programa se ejecutan en el mismo espacio de memoria.
b) Los hilos pueden compartir la pila de ejecución en sistemas distribuidos.
c) En los hilos se primó el consumo frente a la independencia.
d) Un fallo en un hilo puede provocar una finalización inesperada del programa.

Pregunta 1.- Seleccione la respuesta FALSA respecto a la programación concurrente

a) La programación concurrente NO permite ofrecer un entorno interactivo a múltiples usuarios.

b) Los procesos se pueden comunicar entre sí, pero son unidades autónomas e independientes.

c) En un programa secuencial las instrucciones deben ejecutarse una a continuación de otra siguiendo una secuencia determinada por un algoritmo.

d) La programación concurrente se ocupa del estudio y desarrollo de programas que puedan realizar varias tareas "al mismo tiempo".

Pregunta 4.- Señale cuál de las siguientes afirmaciones es FALSA sobre la ejecución de procesos:

a) La programación concurrente surgió para aprovechar el procesador ejecutando varios procesos de forma concurrente.

b) Inicialmente la concurrencia sólo la usaba el sistema operativo para la ejecución de varios programas de forma simultánea, pero internamente los programas no podían realizar varias tareas concurrentemente.

c) La programación concurrente nos permite aprovechar el procesador cuando un proceso espera por una operación de entrada/salida.

d) Los programas secuenciales NO pueden iniciarse por segunda vez antes de haber finalizado la ejecución previa.

Pregunta 4.- Comparando procesos frente a hilos, escoja la respuesta CORRECTA:

a) Los hilos son más robustos que los procesos.

b) Los hilos tardan más tiempo en iniciarse que los procesos.

c) Los hilos comparten el espacio de memoria con los procesos.

d) Ninguna de las anteriores es correcta

Pregunta 1.- Seleccione la respuesta VERDADERA respecto a procesos:

a) Los primeros programas concurrentes estaban formados por varios procesos colaborando entre sí.

b) Los procesos pueden compartir el espacio de memoria.

c) Es representado en el disco duro como un fichero ejecutable.

d) Ninguna de las respuestas anteriores es correcta.

Pregunta 1.- Selecciona la opción que describe correctamente el concepto de "memoria compartida" en programación concurrente.

a) Es una técnica de programación que permite a los hilos acceder a regiones de memoria específicas sin restricciones de acceso o sincronización.

b) Es un mecanismo utilizado para asignar de manera equitativa el espacio de memoria entre diferentes hilos en un sistema multi-hilo.

c) Es una característica de los lenguajes de programación concurrente que permite el uso compartido de la memoria caché entre múltiples hilos para mejorar el rendimiento del programa.

d) Es un enfoque de programación en el que los hilos se comunican e intercambian datos mediante el uso de variables compartidas en el heap.

Pregunta 2.- ¿Cuál de las siguientes afirmaciones sobre los procesos en un sistema operativo es correcta?

a) Los procesos son unidades autónomas e independientes.

b) Los procesos comparten el mismo espacio de memoria.

c) Los procesos son dependientes entre sí.

d) Los procesos son gestionados por las aplicaciones.

2.¿Dónde se usa la programación concurrente?

Enumera los distintos tipos de arquitecturas concurrentes y explícalos por medio de un gráfico

Encontramos varios tipos de arquitecturas de sistemas:

- Monoprocesador: hay un único core en un único chip, son las arquitecturas de hace unos años
- Multiprocesadores muy acoplados: hay varios cores en un único chip, es decir, un procesador con varios núcleos. La mayoría de procesadores actuales tienen esta arquitectura.
- Multiprocesadores poco acoplados: o también llamados de memoria distribuida. Tienen un único core por cada chip y se tratan de varios dispositivos conectados a una red para comunicarse entre sí
- Híbridos: varios dispositivos conectados a una red con varios núcleos por cada procesador

Cuál es la diferencia entre paralelismo real y paralelismo simulado. Describe con un ejemplo algún caso en el que la eficiencia del paralelismo real y del paralelismo simulado puedan ser parecidas. (1 punto)

El paralelismo real surge cuando hay un proceso por cada procesador existente y por lo tanto se aumenta la velocidad. Este paralelismo se da en Multiproceso o en Procesamiento Distribuido.

El paralelismo simulado surge cuando hay varios procesos en un mismo procesador y se produce el efecto de paralelismo real. Permite aprovechar el procesador cuando uno de los proceso espera a una operación de E/S. Se da en la Multiprogramación.

Como ejemplo se podría utilizar una aplicación de procesado de gráficos. Se pueden dividir los píxeles de la imagen entre varios procesadores o núcleos de la CPU o entre distintos hilos de un solo procesador. Si el Software y Hardware están bien optimizados la efectividad de ambos enfoques sería muy parecida. Aunque en el enfoque de los varios procesadores se notaría una pequeña mejora debido a que cada procesador podría dedicar mayor número de recursos de procesado a la tarea a realizar.

Pregunta 2.- ¿Qué técnica es propia del paralelismo simulado?

a) En multiproceso.
b) En multiprogramación.
c) En procesamiento distribuido.
d) Ninguna de las respuestas anteriores es correcta.

Pregunta 3.- Se denomina sistema de memoria distribuida a:

a) Sistema monoprocesador.
b) Sistema multiprocesador poco acoplado.
c) Sistema multiprocesador muy acoplado.
d) Ninguna de las respuestas anteriores es correcta.

Pregunta 1.- Seleccione la opción VERDADERA

a) Un sistema monoprocesador tiene varios procesadores.
b) Un sistema multiprocesador muy acoplado puede estar integrado por varios procesadores en un solo chip.
c) Un procesador multicore está integrado por varios hilos internos.
d) Ninguna de las anteriores es correcta

Pregunta 4.- Selecciona la respuesta CORRECTA en relación con la relevancia de la Ley de Moore en programación concurrente

a) La Ley de Moore establece que el número de hilos en un programa concurrente debe ser proporcional al tamaño de la memoria disponible en el sistema.
b) La Ley de Moore establece que los procesadores actuales son incapaces de ejecutar programas concurrentes de manera eficiente.
c) La Ley de Moore indica que el número de núcleos en los procesadores modernos está limitado y no puede soportar la ejecución concurrente de múltiples hilos.
d) Ninguna de las anteriores

TEMA 2: Modelos de concurrencia

TEMA 2.2 Memoria compartida

1.Intercalación de instrucciones: indeterminismo

¿Cuáles son las 3 abstracciones de la programación concurrente?

- Primera abstracción: se considera que cada proceso se ejecuta en su propio procesador
- Segunda abstracción: se ignoran las velocidades relativas de los procesos. Esto permite considerar únicamente las secuencias de instrucciones ejecutadas.
- Tercera abstracción:considerar que las secuencias de ejecución de las instrucciones atómicas se intercalan en una única secuencia.

Enuncia y explica la 3ª Abstracción de la programación concurrente

Se considera que las secuencias de ejecución de instrucciones atómicas se intercalan en una única secuencia. De esta forma no hay solapamientos y por lo tanto, la ejecución de instrucciones atómicas en paralelo o en secuencial sería exactamente igual. De modo que el resultado de varias ejecuciones siempre será el mismo.

¿Qué es una instrucción atómica? Enumera y describe los tipos de instrucciones atómicas

Es aquella instrucción cuya ejecución es indivisible, es decir, a la hora de ejecutarse no permite interferencias ni otros procesos pueden interferir en ella.

Tipos de instrucciones atómicas:

- Grano fino: instrucciones máquina del procesador
- Grano grueso: sentencias que se ejecutan en un proceso sin interferencias

Ben-Ari propuso en su libro "Principles of Concurrent and Distributed Programming" tres abstracciones para facilitar la programación concurrente y evitar pensar en la arquitectura del sistema. Enuncia las tres abstracciones y desarrolla una de ellas. (2 puntos)

- Primera abstracción: se considera que cada proceso se ejecuta en su propio procesador
- Segunda abstracción: se ignoran las velocidades relativas de los procesos. Esto permite considerar únicamente las secuencias de instrucciones ejecutadas.
- Tercera abstracción:considerar que las secuencias de ejecución de las instrucciones atómicas se intercalan en una única secuencia. De esta forma no hay solapamientos y por lo tanto, la ejecución de instrucciones atómicas en paralelo o en secuencial sería exactamente igual. De modo que el resultado de varias ejecuciones siempre será el mismo.

Pregunta 2.- Señale cuál de las siguientes afirmaciones es CIERTA respecto a las instrucciones atómicas:

a) Su ejecución es divisible.
b) Otros procesos pueden interferir en su ejecución.
c) Toda sentencia de un lenguaje es atómica.
d) Ninguna de las respuestas anteriores es correcta.

Pregunta 4.- ¿Cuál es una consideración importante al diseñar programas concurrentes en términos de la velocidad relativa de los procesos?

a) Evaluar los posibles efectos de las velocidades relativas en el comportamiento del programa.
b) Asegurarse de que todos los procesos tengan la misma velocidad para evitar problemas.
c) Ignorar por completo las velocidades relativas para simplificar el diseño.
d) Asignar un procesador más rápido a los procesos críticos para mejorar el rendimiento.

3.Sincronización con espera activa

¿Qué es la exclusión mutua? ¿Por qué se utiliza esta técnica en programación concurrente? ¿Qué ocurriría si no se utilizara?

La exclusión mutua ocurre cuando varios procesos compiten por el acceso a un recurso compartido y solo uno de ellos puede acceder a él a la vez. Es necesaria en programación concurrente para el tratamiento de variables comunes a varios procesos o hilos.

Debido a esto, es necesario que secuencias de instrucciones que se vean relacionadas con el recurso compartido se ejecuten sin intercalaciones y solo por un proceso al mismo tiempo.

Si no se utilizara la exclusión mutua los recursos compartidos se modificarían de forma diferente en cada ejecución del código provocando resultados distintos cada vez.

Explica qué hace volatile y la diferencia o relación con la exclusión mutua

Volatile se utiliza para aquellas variables compartidas entre varios procesos o hilos. La exclusión mutua se intentó implementar mediante variables de tipo volatile, es decir, compartidas pero estas provocan interbloqueos o permitían que varios procesos ejecutan su sección de exclusión mutua a la vez (las instrucciones se intercalaban). Es por eso que para la realización de exclusión mutua no es aconsejable el uso de este tipo de variables, es aconsejable el uso de herramientas de sincronización.

4. Propiedades de corrección

Elige una propiedad de vida de las propiedades de corrección y escribe un ejemplo que explique su comportamiento.

La propiedad elegida ha sido ausencia de interbloqueos activos. Los interbloqueos activos (livelock) ocurren cuando hay instrucciones que impiden progresar a los procesos y que se usan para sincronizarlos entre sí.

Por ejemplo, cuando tenemos una exclusión mutua realizada con un mutex y contamos con dos procesos que quieren acceder a una sección crítica donde se aumenta una variable contador. Los procesos tendrán una instrucción de mutex.Wait() antes de empezar la sección crítica. En caso de que uno de los dos procesos ya esté dentro, el otro proceso no avanzará hasta que el mutex se libere. Durante ese tiempo de espera se produce un interbloqueo activo.

¿Qué es una condición de carrera?

Ocurre cuando el resultado de un programa depende del orden en el que se ejecutan las operaciones en múltiples hilo. Una condición de carrera se convierte en un error cuando alguno de los comportamientos posibles es indeseable.

¿Qué son las propiedades de corrección en Memoria compartida? Enumera también la clasificación general y los tipos de propiedades de corrección, escoge uno de ellos y pon un código de ejemplo. No está permitido el uso de SimpleConcurrent ni espera activa.

Son propiedades que se deben cumplir en un programa de Memoria compartida para que sea correcto.

Hay dos tipos de propiedades de corrección:

- De seguridad: si no se cumplen estas propiedades el programa se comportará de forma incorrecta
    - Ausencia de interbloqueos pasivos (deadlock)
    - Exclusión mutua para los recursos compartidos
- De vida: si no se cumplen estas propiedades el programa se comportará de forma correcta pero será más lento y desaprovechará los recursos
    - Ausencia de retrasos innecesarios
    - Ausencia de inanición
    - Ausencia de interbloqueos activos (livelock)

Ejemplo de la ausencia de interbloqueos activos: código con semáforos o mutex.

¿Cuáles son los dos problemas principales a los que debe hacer frente un programa concurrente?

Las condiciones de carrera, es decir, cuando el resultado de un programa depende del orden en el que se ejecutan sus instrucciones dando lugar a resultados indeseados.

Los interbloqueos pasivos (deadlock) que se dan cuando uno o varios procesos esperan indefinidamente

Explica en qué consiste la propiedad de corrección de ausencia de inanición y pon un código de ejemplo en el que ocurra dicho problema

La ausencia de inanición permite que todo proceso que quiera acceder a un recurso compartido puede hacerlo en algún momento.

```
static Mutex mutex = new Mutex();
static volatile int compartida=0;
static void proceso1(){
      mutex.WaitOne();
      compartida++;
}
static void proceso2(){
      mutex.WaitOne();
      compartida--;
      mutex.Release();
}
private static void Main(){
      new Thread(proceso1).Start();
      new Thread(proceso2).Start();
}
```

En este caso el proceso 2 nunca podrá acceder a la variable compartida ya que el primer proceso nunca libera el mutex.

Enumera las propiedades de corrección que debería tener un programa concurrente. Elige una de ellas y explícala brevemente. (1.5 puntos)

Exclusión mutua para el tratamiento de recursos compartidos

Ausencia de interbloqueos:

- Activos: cuando un instrucción no permite el avance del programa ya que sirven para sincronizarlo con otros
- Pasivos: cuando dos o varios procesos esperan indefinidamente

Ausencia de retrasos innecesarios: que el programa progrese en su ejecución

Ausencia de Inanición: que todo proceso que quiera acceder a un recurso compartido pueda hacerlo.

En qué dos grupos se clasifican las propiedades de corrección de los programas concurrentes. Para cada uno de ellos, describe una situación práctica en el que se incumpla una propiedad de corrección, y explica qué consecuencias tiene. (1.5 puntos)

Hay dos tipos de propiedades de corrección:

- De seguridad: si no se cumplen estas propiedades el programa se comportará de forma incorrecta
    - Exclusión mutua para los recursos compartidos

```
static Mutex mutex = new Mutex();
static volatile int compartida=10;
static void proceso1(){
      while(compartida>0)
            compartida++;
}
static void proceso2(){
      while(compartida>0)
            compartida= compartida -2;
}
private static void Main(){
      new Thread(proceso1).Start();
      new Thread(proceso2).Start();
}
```

En este caso se accede al recurso compartido sin ningún tipo de cuidado y por lo tanto el resultado de la ejecución del código no será siempre el mismo, generando comportamientos indeseados

- De vida: si no se cumplen estas propiedades el programa se comportará de forma correcta pero será más lento y desaprovechará los recursos
    - Ausencia de inanición

```
static Mutex mutex = new Mutex();
static volatile int compartida=0;
static void proceso1(){
      mutex.WaitOne();
      compartida++;
}
static void proceso2(){
      mutex.WaitOne();
      compartida--;
      mutex.Release();
}
private static void Main(){
      new Thread(proceso1).Start();
      new Thread(proceso2).Start();
}
```

En este fragmento el proceso dos jamás podrá acceder al recurso compartido ya que el proceso 1 nunca libera el mutex.

6.Introducción a semáforos

¿Cómo se puede implementar la exclusión mutua con un semáforo? Pon código de ejemplo

La exclusión mutua con un semáforo se implementa dándole al semáforo un único permiso. Este permiso es utilizado por el proceso que quiere acceder a un dato compartido y tras acceder a él, lo libera para que acceda otro proceso.

```
SlimSemaphore sem;
static volatile int compartida=0;
static void proceso1(){
      sem.Wait();
      compartida++;
      sem.Release();
}
static void proceso2(){
      sem.Wait();
      compartida--;
      sem.Release();
}
private static void Main(){
      sem = new SlimSemaphore(1);
      new Thread(proceso1).Start();
      new Thread(proceso2).Start();
}
```

¿Cómo se implementa la sincronización condicional con un semáforo? Pon código de ejemplo

Para implementar la sincronización condicional con un semáforo, el semáforo tiene que tener cero permisos iniciales. El permiso se otorga cuando se cumple la condición, hasta ese entonces los procesos esperan.

```
SlimSemaphore sem;
static volatile int compartida=0;
static void proceso1(){
      sem.Wait();
      Console.WriteLine("Se ha llegado a 5 ticks");
}
static void proceso2(){
      while(comopartida<5){
```

```
                compartida++;
                }
                sem.Release();
        }
        private static void Main(){
                sem = new SlimSemaphore(1);
                new Thread(proceso1).Start();
                new Thread(proceso2).Start();
        }
```

Pregunta 9.- Dados dos procesos:

```
public void proceso1() {
        progress = progress + 20;
        if (progress >= 20) {
                semaphore.release(); }
        semaphore.acquire();
        }
public void proceso2() {
        progress = progress + 30;
        if (progress >= 30) {
                semaphore.release(); }
        progress = progress + 50;
         if (progress >= 80) {
                semaphore.release(); }
        semaphore.acquire();
        }
```

a) ¿Los procesos pueden tener problemas de concurrencia? Justifica tu respuesta.

Sí ya que se está accediendo a la variable compartida progress sin tener una sincronización adecuada. Esto puede producir una condición de carrera ya que el resultado de la ejecución dependerá de la velocidad de los distintos procesos.

b) El código en dichos procesos está incompleto. ¿Qué falta y por qué es necesario?

c) ¿Cómo modificarías el código de proceso1 y proceso2 para evitar errores?

```
public void proceso1() {
        semaphore.Wait();
        progress = progress + 20;
        if (progress >= 20) {
                semaphore.Release(); }
        }
public void proceso2() {
        semaphore.Wait();
        progress = progress + 30;
        if (progress >= 30) {
                semaphore.release(); }
        progress = progress + 50;
         if (progress >= 80) {
                semaphore.Release(); } }
```

Pregunta 5.- Dado el siguiente código, explica cómo y dónde se puede dar deadlock. Justifica teóricamente la argumentación. (1.5 puntos)

```
const int NProcesos = 3;
static volatile int nProcesos;
static SemaphoreSlim barrier;
private static void Proceso(){
        WriteLine("A");
        nProcesos++;
        if (nProcesos<NProcesos)
                barrier.Wait();
        else
```

```
                for (int i = 0; i < NProcesos-1; i++)
                      barrier.Release();
          WriteLine("B");
    }
    public static void Main(String[] args){
          nProcesos = 0;
          barrier = new SemaphoreSlim(0);
          for (int i = 0; i < NProcesos; i++)
                new Thread(Proceso);}
```

Debido a que la variable compartida nProcesos no está protegida correctamente se puede dar el caso en el que dos de los hilos aumentan la variable justo al mismo tiempo. Esto hará que tras el proceso 1 y el 2 la variable nProcesos valga 1. Cuando el proceso 3 aumente nProcesos (a 2) nunca entrará en el else y nunca se desbloquearán al resto de procesos. Se debería encerrar el aumento de la variable compartida bajo una exclusión mutua.

7.Sincronización avanzada

¿Qué es un esquema productor-consumidor en programación concurrente? Cuando se implementa con buffer, ¿qué requisitos tiene que cumplir el buffer?

Es un problema de programación concurrente en el que los Productores producen datos a la vez que los Consumidores los consumen. Los Productores solo generan un dato a la vez y los Consumidores consumen un dato a la vez. Ningún dato puede consumirse dos veces y todos los datos deben ser consumidos. Los datos generados se almacenan temporalmente en un array que tiene que ser circular ya que es más rápido a la hora de acceder a los datos y más eficiente en cuanto al uso de memoria.
Los Productores deben bloquearse cuando no queden huecos para producir más datos, es decir, cuando el array está vacío. Los Consumidores tienen que bloquearse cuando no tengan datos que leer, es decir, el array está vacío.

TEMA 2.3: Paso de mensajes

¿Qué son TCP y UDP? ¿En qué se diferencian? ¿Por qué sus diferencias son importantes para los juegos online?

Son protocolos de envío de mensajes desde un dispositivo a otro.

- En TCP se utiliza un buffer para almacenar los mensajes enviados por el emisor y desde donde los lee el receptor, sin embargo, en UDP no existe este buffer.
- UDP es un protocolo de mínimo esfuerzo, es decir, la entrega de los mensajes no está garantizada mientras que en TCP sí.
    - Por ello, los paquetes de TCP son más grandes ya que contienen la información de pérdida de paquetes.
    - También por esta razón el protocolo UDP tiene menos latencia puesto que no es necesaria la verificación.
- En UDP no existen ni clientes ni servidores por lo tanto el cliente no necesita escuchar un puerto.
- TCP es un protocolo orientado a la conexión mientras que UDP no lo es.
- Además, en UDP no hay garantía de orden y en TCP sí.

En juegos online se requiere una latencia baja y UDP al no garantizar entrega fiable nos la proporciona. También es necesario perder la mínima cantidad de datos posibles, para esto se utilizará TCP.

¿Qué protocolos de comunicación se pueden usar al implementar una aplicación en un navegador web? ¿En qué se parecen y diferencian?

Se pueden utilizar dos protocolos TCP y UDP.

- En TCP se utiliza un buffer para almacenar los mensajes enviados por el emisor y desde donde los lee el receptor, sin embargo, en UDP no existe este buffer.
- UDP es un protocolo de mínimo esfuerzo, es decir, la entrega de los mensajes no está garantizada mientras que en TCP sí.

    - Por ello, los paquetes de TCP son más grandes ya que contienen la información de pérdida de paquetes.
    - También por esta razón el protocolo UDP tiene menos latencia puesto que no es necesaria la verificación.
- En UDP no existen ni clientes ni servidores por lo tanto el cliente no necesita escuchar un puerto.
- TCP es un protocolo orientado a la conexión mientras que UDP no lo es.
- Además, en UDP no hay garantía de orden y en TCP sí.

Pregunta 4.- Existen muchos tipos concretos de modelos de paso de mensajes dependiendo de:

a) Identificación de procesos.
b) Sincronización.
c) Características del canal.
d) Todas las respuestas anteriores son correctas.

Pregunta 4.- Seleccione la opción FALSA, en relación con el modelo de paso de mensajes:

a) Los procesos solamente se pueden comunicar mediante el envío de mensajes
b) No pueden usarse en sistemas muy acoplados.
c) La comunicación directa simétrica implica que tanto el emisor como el receptor indican el id del otro proceso.
d) En la comunicación asíncrona, emisor y receptor no tienen porqué coincidir en el tiempo para llevar a cabo la comunicación

Pregunta 1.- ¿Qué recursos son necesarios para que dos procesos puedan comunicarse entre sí utilizando sockets de Internet?

a) Una dirección IP y un número de puerto.
b) Un par de direcciones MAC.
c) Un nombre de dominio y una dirección IP.
d) Un número de puerto y un nombre de archivo.

TEMA 3: Concurrencia en los lenguajes

Enfoques para la concurrencia en C++.

Hay tres enfoques:

- Concurrencia en el lenguaje: realizado mediante palabras reservadas que permiten la paralelización del código
    - Se utiliza OpenMp que sigue el esquema de fork/join y Intel Cilk Plus que añade tres palabras reservadas para paralelizar sobre arrays.
- Concurrencia con librerías de terceros: con Windows API y POSIX que permiten un modelos de memoria compartida o con librerías multiplataforma que utilizan herramientas de alto nivel
- Concurrencia con librerías estándar: que llegar en el C++11

¿Qué es un Web Worker? ¿Qué modelos de concurrencia utiliza? ¿Cuál es su funcionamiento?

Es una característica del lenguaje de programación en JavaScript que permite realizar el modelo concurrente de paso de mensajes. Se le llama Web Worker o Worker a los hilos que se ejecutan en segundo plano. Este ejecuta código ante la llegada de un mensaje desde el script principal u otro Worker.

Pregunta 2.- ¿Cuál de los siguientes no es un enfoque para programación concurrente en C/C++?

a) Concurrencia con librerías de terceros
b) Concurrencia con la librería estándar
c) Extensiones del Sistema Operativo
d) Extensiones de compiladores

Pregunta 2.- Seleccione la opción FALSA

a) Java ofrece un modelo de concurrencia de memoria compartida
b) En C/C++ existen extensiones de algunos compiladores que permiten la paralelización del código.
c) JavaScript permite un modelo de concurrencia de paso de mensajes.
d) En C/C++, la biblioteca estándar no dispone de un modelo de memoria compartida.

TEMA 4: Aspectos Avanzados

TEMA 4.1: Expresiones lambda

¿Qué líneas de código se deben insertar en los marcadores A y B para que el evento OnEnemyDeath se invoque correctamente cuando el enemigo muera y se ejecute el método OnEnemyDeath() de la clase Game?

```
public class Enemy {
	public event Action OnEnemyDeath;
	private void Die() {
		enemy.OnEnemyDeath?.Invoke();
	}
}
public class Game {
	private Enemy enemy;
	public void Start() {
		enemy = new Enemy();
		enemy.OnEnemyDeath += OnEnemyDeath;
	}
	private void OnEnemyDeath() {
		Console.WriteLine("El enemigo ha muerto.");
	}
}
```

enemy.OnEnemyDeath += OnEnemyDeath;
enemy.OnEnemyDeath?.Invoke();
enemy.Die();

OnEnemyDeath += OnEnemyDeath;
enemy.OnEnemyDeath - OnEnemyDeath;
OnEnemyDeath();

TEMA 4.2: Gestión de hilos en C#

A veces es necesario finalizar la ejecución de un hilo antes de que acabe por sí mismo. Explica por qué no es conveniente que un hilo pueda forzar la detención de otro, y detalla la forma correcta de resolver este problema. (2 puntos)

Cuando un hilo finaliza abruptamente sus tareas quedan a medias y algunos objetos pueden quedar en un estado inconsistente. Por ello es necesario que se pregunte constantemente si el hilo ha sido interrumpido y de esta forma poder finalizarlo correctamente (liberando recursos, cerrando conexiones y dejando los objetos en un estado estable). Cuando se interrumpe un hilo hay que poner un Thread.Join() para esperar a que termine de finalizar antes de continuar.

Los hilos interrumpidos levantan la excepción ThreadInterruptedException cuando otro hilo ejecuta Thread.Interrupt(). Esta excepción no se debe ignorar, el hilo debe finalizar de forma controlada

Pregunta 5.- Explica brevemente la función de las memory barriers en C# y describe su relevancia en la programación concurrente. (1 punto)

Una memory barrier es un sistema que asegura que un hilo lee el último valor de un atributo compartido. Es importante ya que si los atributos compartidos no están marcados, a la hora de ejecutar, se reorganizan las lecturas y escrituras para obtener un mejor rendimiento y esto puede hacer que se lean/escriban valores erróneos provocando condiciones de carrera.

¿En qué casos en java se puede compartir un objeto entre hilos sin que ocurran problemas de concurrencia?

- Es un objeto de tipo Thread-safe
- Si se accede a ese objeto bajo una exclusión mutua
- Si el objeto no va a ser modificado mientras se comparte entre hilos
- La clase a la que pertenece es inmutable, es decir, nunca cambia de estado y solo puede ser establecido en la inicialización

Pregunta 4.- En qué caso no es seguro compartir objetos entre hilos:

a) El objeto es inmutable.

b) Se accede al objeto bajo exclusión mutua.

c) El objeto es volatile.

d) El objeto es thread-safe.

Pregunta 2.- ¿En qué caso NO es seguro compartir un objeto entre varios hilos en c#?

a) El objeto es inmutable.

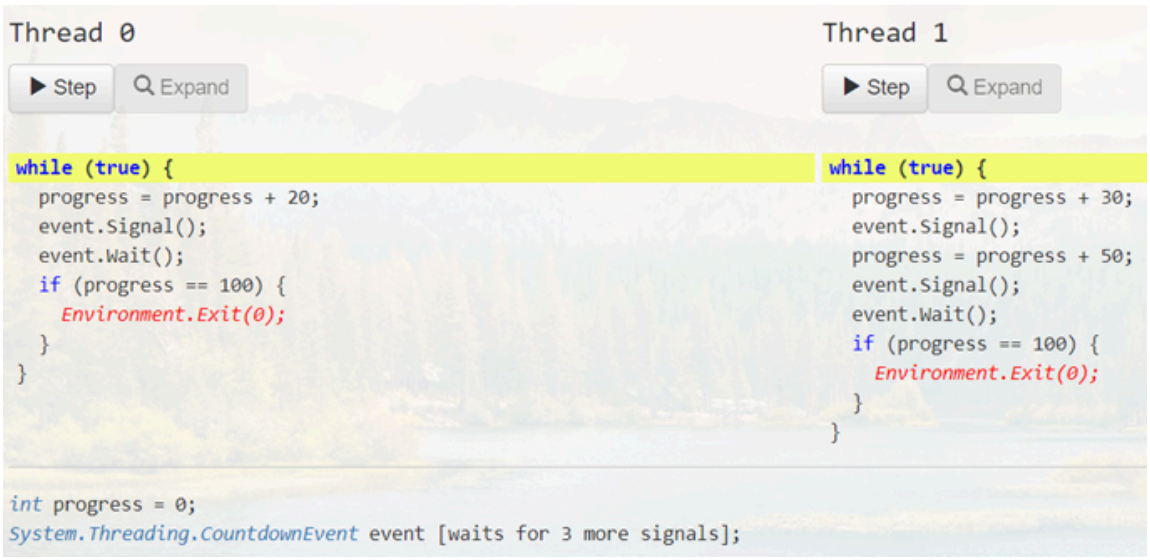
b) El objeto no se modifica mientras se comparte entre los hilos.

c) El objeto está marcado como volatile.

d) Los métodos del objeto están preparados para su ejecución concurrente.

TEMA 4.3: Sincronización de hilos

Analiza el siguiente código y explica en qué circunstancias se podría llegar a un deadlock.



Ojo a Environment.Exit(0), que es un método que finaliza con éxito la ejecución del programa.

TEMA 4.4: Estructuras de datos concurrentes

Explica las diferencias entre las estructuras de datos concurrentes y sincronizadas

Las estructuras de datos concurrentes están diseñadas específicamente para permitir el acceso y la modificación de datos de forma concurrente entre hilos.

Las estructuras sincronizadas son las que utilizan mecanismos de sincronización como bloqueos para controlar el acceso concurrente y garantizar la consistencia.

Usando estructuras concurrentes se usan bloqueos más ligeros que las estructuras sincronizadas y son más eficientes en entornos con mucha concurrencia.

Las estructuras sincronizadas son más fáciles de implementar y entender que las concurrentes

Pregunta 6.- En el contexto de las estructuras de datos concurrentes, como las utilizadas en el desarrollo de videojuegos, los conceptos de dirty reads y snapshots son de suma importancia, ya que sirven para poder recorrer de manera segura una estructura de datos concurrente.. Explica qué se entiende por "dirty reads" y cómo pueden ocurrir en un entorno concurrente. Además, explique qué son los "snapshots" y cómo se utilizan para evitar los dirty reads en las operaciones concurrentes en las estructuras de datos. (1 punto)

Los dirty reads ocurren cuando se recorren estructuras de datos que se modifican de forma dinámica. Esto provoca que se pueden leer datos antiguos y datos nuevos. En un entorno concurrente se da cuando hay estructuras de datos siendo modificadas por varios hilos a la vez.
Los snapshots son una copia instantánea de una estructura de datos. Esto se realiza para que a la hora de leer no se puedan modificar los datos y se obtenga una lectura consistente.

¿Cuál de las siguientes estructuras de datos en C# es una colección thread-safe para elementos no ordenados y que permite duplicados?

a. ProducerConsumerCollection<T>
b. ConcurrentQueue<T>
c. ConcurrentMap<K, V>
d. ConcurrentBag<T>

TEMA 4.5: Ejecución de tareas

Describe la problemática de utilizar la estrategia de crear un hilo por petición en entornos de producción. ¿Cuáles son las soluciones propuestas?

Crear un hilo es un proceso costoso y cada hilo ocupa memoria (para la pila de ejecución) y tiempo de CPU (para realizar los cambios de contexto). Por eso,creando hilos por petición se podría llegar al número máximo de hilos permitidos y esto afectaría al sistema. También, en ese caso, se consumirían demasiados recursos de memoria y CPU.
Para ello se utiliza la clase ThreadPool() que proporciona un conjunto de subprocesos para la ejecución de tareas. De esta forma se permite reutilizar los hilos (evitando el coste de creación/destrucción) y se puede especificar un número máximo de hilos para que no llegue a afectar al sistema.

Describe los problemas de crear un hilo por cada petición en un sistema en producción con mucha carga

Crear un hilo es un proceso costoso y cada hilo ocupa memoria (para la pila de ejecución) y tiempo de CPU (para realizar los cambios de contexto). Por eso, en un sistema con mucha carga serían necesarios muchos hilos. De esta forma se podría llegar al número máximo de hilos permitidos y esto afectaría al sistema. También se consumiría demasiados recursos de memoria y CPU.

¿Qué es el thread pool y qué ventajas tiene su uso con respecto a la creación manual de hilos? (1 punto)

El thread pool es una clase que proporciona un conjunto de subprocesos para ejecutar tareas.
Crear hilos de forma manual es costoso, además se puede llegar al número máximo de hilos que se permiten y que afecte al sistema.
Mediante el thread pool se permite la reutilización de los hilos para ahorrarse el coste de creación/destrucción de los hilos de forma manual. Además esta clase permite especificar un número máximo de hilos para que así no llegue a afectar al sistema.

Los threads son una forma de concurrencia de bajo nivel, y por ello tienen ciertas limitaciones. Enumera y describe brevemente estas limitaciones. (1.5 puntos)

No hay una forma sencilla de devolver valores en los hilos. Esto se debe hacer con variables compartidas y exclusión mutua.
La propagación de errores entre hilos es complicada. Si ocurre un error en un hilo, afecta al resto.
No se puede especificar que el hilo haga otra cosa cuando ya ha terminado su tarea.
Muchos hilos con operaciones de E/S acaban degradando el sistema.
Además, cada hilo ocupa espacio de memoria (para la pila de ejecución) y ciclos de CPU (para el cambio de contexto) por lo que si hay un número elevado de ellos se consumen muchos recursos.

Pregunta 3.- ¿Cuál de las siguientes afirmaciones es falsa?

a) Es recomendable utilizar el ThreadPool con tareas de larga duración.
b) Una desventaja de usar threads en C# es que no hay forma sencilla de devolver valores desde el thread.
c) Tareas pequeñas favorecen la escalabilidad y el balanceo de carga.
d) El ThreadPool permite reutilización de hilos para varias tareas

¿Cuál de las siguientes afirmaciones es una ventaja de usar el ThreadPool?

a. El ThreadPool requiere que se cree un nuevo hilo para cada tarea nueva.
b. El ThreadPool impide que se ejecuten tareas de manera concurrente.
c. El ThreadPool permite la reutilización de hilos para varias tareas, evitando la necesidad de crear un hilo por cada tarea.
d. El ThreadPool no permite especificar el orden de ejecución de las tareas.

TEMA 5: Introducción a los sistemas distribuidos

En 1997, Ensemble Studios desarrolla el icónico juego Age of Empires. A nivel de multijugador, el juego era capaz de gestionar cientos de unidades de sus 8 jugadores a pesar de las conexiones lentas de la época, gracias a la estrategia deterministic lockstep. Enuncia las particularidades de esta estrategia y describe su funcionamiento. (1 punto)

La estrategia de deterministic lockstep consistía en pasar a cada jugador únicamente su información de control y no la de los otros jugadores. Con esta información cada jugador realizaba una simulación determinista de cada unidad. Es decir, con el mismo estado inicial y las mismas entradas todas las unidades realizaban la misma simulación. Esto hacía posible que cada jugador pudiera mover muchísimas unidades al mismo tiempo.

Analiza el impacto de Doom en la evolución técnica del género de juegos multijugador y describe cómo Quake superó las barreras técnicas y de diseño establecidas por su predecesor

Doom es un FPS para cuatro jugadores. Fue tan sonado ya que utilizó el protocolo privativo de IPX sobre LAN y una arquitectura Peer-To-Peer donde todos los jugadores se conectaban a la misma red.
Se utilizó la técnica del broadcasting que consistía en que toda la información se pasaba a todos los jugadores. Esto tuvo el problema de que cuando había muchos paquetes llegando a los clientes, muchos de ellos se perdían.
Además, cuando había más de 5 partidas simultáneas la red se saturaba y no permitía a los jugadores jugar. Esto ocurrió ya que el equipo de desarrollo no se esperaba la acogida del juego y únicamente lo probaron con su propia partida a la vez.
En el juego Quake se habilitaron servidores 24 horas con IPs públicas y se utilizó la arquitectura Cliente-Servidor. Fue el primer juego en el que los jugadores no tenían que conocer al resto ni quedar a una determinada hora para poder jugar.
Debido a la existencia de estos servidores surgieron nuevos términos en cuanto a juegos multijugador como la latencia ya que las conexiones al servidor de algunos jugadores podrían ser lentas y causar ping.
Más tarde se realizó una actualización gratuita llamada QuakeWorld donde se mejoraron las comunicaciones y se utilizaba la técnica del client-side prediction. Esta técnica consistía en predecir parcialmente lo que iba a realizar el jugador para conseguir un juego más fluido aunque no terminó con la latencia.

TEMA 6: Aspectos de Diseño

1.Arquitecturas de juegos

Describe las diferentes topologías para desarrollar un juego en red explicando sus ventajas e inconvenientes

Arquitectura Peer-To-Peer: es una red entre pares. Los clientes se conectan entre sí sin mediadores y cada jugador es responsable de ciertos elementos. Hay dos tipos de clientes en esta arquitectura: los heterogéneos (algunos clientes tienen más prioridad que otros) y los homogéneos.

- Ventajas:
  - La retransmisión de los datos es más rápida ya que no tiene que llegar a un servidor y volver

        - Los costes son económicos ya que no hay que implementar un servidor
        - Tiene una mayor fiabilidad puesto que no hay un único punto de fallo, es decir, si uno de los clientes falla el resto puede continuar
        - Es fácil de escalar ya que cada vez que se añade un jugador, éste aporta sus recursos
    - Desventajas:
        - Es una arquitectura menos segura ya que es más fácil hacer trampas
        - Es difícil de gestionar puesto que no hay un punto con toda la información global
        - Es difícil mantener la consistencia ya que las correcciones se deben hacer en distintos puntos
        - Más complejo de implementar

Arquitectura Cliente-Servidor: hay un servidor centralizado que conecta los distintos clientes. Es la arquitectura más popular actualmente.

- Ventajas:
    - La integración es simple puesto que contamos con un punto que contiene toda la información global
    - Hay una mayor fiabilidad ya que los datos que provienen del servidor son correctos
    - Los clientes tienen menos carga de trabajo y son más sencillos
    - Es más sencillo de implementar
- Desventajas:
    - Es más costoso ya que hay que implementar un servidor y mantenerlo
    - No es escalable ya que si el servidor llega a su máxima capacidad de clientes no se puede hacer nada al respecto
    - Hay un único punto de fallo, el servidor. Si este falla lo hacen todos los clientes

Arquitectura Listen Server: es una variante del Cliente-Servidor dónde uno de los clientes es también servidor y es conocido como host

- Ventajas:
    - Las ya mencionadas antes en la arquitectura Cliente-Servidor
    - Los jugadores que se encuentran en la misma región que el host tienen menos lag
- Desventajas:
    - Las ya mencionadas antes en la arquitectura Cliente-Servidor
    - Se puede dar latencia y pérdida de paquetes ya que se depende de la red local del host que en ocasiones puede ser mala o estar congestionada

Arquitectura híbrida: combina Peer-To-Peer y Cliente-Servidor. En el servidor se almacena la información global y se solucionan conflictos y la información menos importante se mueve con Peer-To-Peer

- Ventajas:
    - La latencia se reduce ya que en zonas de latencia crítica los clientes pueden comunicarse directamente con P2P.
    - Hay mayor tolerancia a fallos ya que si el servidor falla la arquitectura P2P puede mantenerlo
    - Es menos costoso puesto que como se depende menos del servidor se puede reducir la inversión en él
- Desventajas:
    - Es complejo de implementar las dos arquitecturas a la vez
    - Mantener la consistencia entre nodos P2P y el servidor es complejo

Red de servidores: consiste en una arquitectura con más de un servidor.

- Hay dos categorías de redes:
    - Juego dividido en instancias completas llamadas shards
    - Juego dividido en regiones geográficas
- Ventajas:
    - Es una arquitectura escalable ya que con la integración de un nuevo servidor se pueden añadir más jugadores

- Es fiable ya que los servidores pueden actuar de backups de otros
- Desventajas:
  - En las redes divididas en shards se limita la interoperabilidad
  - En las redes divididas en regiones es complejo mantener la consistencia al cambiar de región
  - Los costes son muy elevados debido al gran número de servidores

Enumera las diferentes topologías para desarrollar un juego en red y explica las ventajas de una de ellas.

Arquitectura Peer-To-Peer:
- Retransmisión de datos rápida ya que no tiene que ir y volver al servidor
- Costes más económicos ya que no hay que implementar un servidor
- Hay mayor fiabilidad puesto que no existe un único punto de fallo, es decir, si un cliente falla el resto pueden continuar
- Es escalable ya que cada jugador añadido aporta sus recursos

Arquitectura Cliente-Servidor:
- La integración es más fácil ya que en el servidor se encuentra toda la información global
- Los datos que reciben los clientes son fiables ya que provienen del servidor
- Los clientes tienen menos carga de trabajo y son más sencillos
- Es más sencilla de implementar

Arquitectura Listen-Server:
- Las ya mencionadas en la arquitectura anterior pero en este caso el servidor es a su vez un cliente llamado host.
- Los clientes que se encuentran en la misma región que el host tienen menos lag

Arquitectura híbrida:
- En zonas de latencia crítica los clientes se comunican directamente con P2P para reducirla
- Hay una mayor tolerancia a fallos. Si el servidor falla el P2P puede mantenerlo
- Menos costoso ya que al no depender tanto del servidor se puede reducir la inversión en el

Arquitectura de red de servidores:
- Es muy escalable ya que con la integración de un nuevo servidor se pueden añadir jugadores
- Es fiable ya que los servidores pueden actuar de backup entre sí

Describe la arquitectura Cliente-Servidor y enumera y explica brevemente sus ventajas e inconvenientes. (2 puntos)

Arquitectura Cliente-Servidor: hay un servidor centralizado que conecta los distintos clientes. Es la arquitectura más popular actualmente.
- Ventajas:
  - La integración es simple puesto que contamos con un punto que contiene toda la información global
  - Hay una mayor fiabilidad ya que los datos que provienen del servidor son correctos
  - Los clientes tienen menos carga de trabajo y son más sencillos
  - Es más sencillo de implementar que otras arquitecturas
- Desventajas:
  - Es más costoso ya que hay que implementar un servidor y mantenerlo
  - No es escalable ya que si el servidor llega a su máxima capacidad de clientes no se puede hacer nada al respecto
  - Hay un único punto de fallo, el servidor. Si este falla lo hacen todos los clientes

Pregunta 7.- Explica la arquitectura Cliente-Servidor utilizada en los videojuegos, destacando sus características principales. Analiza las ventajas e inconvenientes asociados a esta arquitectura y brinda ejemplos concretos que ilustren su aplicación en el desarrollo de videojuegos. (2 puntos)

Arquitectura Cliente-Servidor: hay un servidor centralizado que conecta los distintos clientes. Es la arquitectura más popular actualmente.
- Ventajas:

    - La integración es simple puesto que contamos con un punto que contiene toda la información global
    - Hay una mayor fiabilidad ya que los datos que provienen del servidor son correctos
    - Los clientes tienen menos carga de trabajo y son más sencillos
    - Es más sencillo de implementar que otras arquitecturas
- Desventajas:
    - Es más costoso ya que hay que implementar un servidor y mantenerlo
    - No es escalable ya que si el servidor llega a su máxima capacidad de clientes no se puede hacer nada al respecto
    - Hay un único punto de fallo, el servidor. Si este falla lo hacen todos los clientes

Pregunta 3.- Seleccione la opción FALSA:

a) En la arquitectura P2P la transmisión de datos es más lenta ya que depende de la velocidad del nodo más lento.

b) La arquitectura Cliente-Servidor facilita la integración de datos al tener una fuente fiable.

c) En las arquitecturas híbridas, la resolución de conflictos no es gestionada por los clientes.

d) En la red de servidores una modalidad consiste en fragmentar el juego en instancias o shards.

Pregunta 3.- Seleccione la opción VERDADERA:

a) En la arquitectura P2P la transmisión de datos es más lenta ya que depende de la velocidad del nodo más lento.

b) La arquitectura Cliente-Servidor complica en exceso la integración de datos al tener múltiples clientes.

c) En las arquitecturas híbridas, la resolución de conflictos es gestionada por los clientes.

d) En la red de servidores una modalidad consiste en fragmentar el juego en instancias o shards.

3. Técnicas de compensación

Enumera las técnicas de compensación y desarrolla dos de ellas

Optimización del protocolo: ya que enviar mensajes tiene un coste de procesado. Para reducirlo se comprimen los mensajes y se reducen el número de estos.

- Comprimir los mensajes consiste en reducir el número de bits del mensaje a la vez que se aumenta el coste computacional de procesado para descomprimirlo. Se realiza con varias técnicas:
    - Comprensión delta: evitar la información redundante
    - Reducir el nivel de detalle de la información enviada
    - Filtrar la información menos relevante. Esta técnica tiene un coste adicional por el filtrado
- Para reducir el número de mensajes también se utilizan dos técnicas:
    - Timeout-based approach: consiste en que la información se manda cada cierto tiempo
    - Quorum-based approach: consiste en que la información se almacena hasta que llega a cierta cantidad para luego enviarse

Interpolación en el cliente: busca reducir la frecuencia de comunicación entre nodos. Para ello se interpola con la información de los dos últimos ticks para conseguir frames intermedios. De esta forma el cliente se actualiza con un framerate mayor al de los ticks del servidor. Hace eljuego más fluido pero no soluciona el lag.

Extrapolación en el cliente: también busca reducir la frecuencia de comunicación entre nodos. Se realiza en dos etapas:

- Predicción: donde se estima el estado nuevo de los objetos
- Convergencia: se comprueba esa predicción con la realidad que proviene del servidor y se realizan las correcciones necesarias.

Predicción en el cliente: consiste en realizar acciones en el cliente nada más obtener el input. Estas acciones realizadas serán iguales a las recibidas por el servidor siempre que no haya una interacción externa. En caso de haberla será necesario hacer la convergencia entre las dos. Implica que se debe duplicar cierta parte de lógica del juego en el cliente.

Interest Manager: esta técnica trata de que el nodo no necesita saber toda la información del juego. El servidor es el que gestiona los objetos de interés del jugador. Con esta técnica se crean tres Game State:

- El del servidor: con el contenido global del juego en el servidor autoritativo
- El publicable: con la información que se sincroniza en un momento determinado
- El del cliente: con la información del cliente particular

Simulación sincronizada: en esta técnica todos los jugadores tienen una copia completa del juego. Las simulaciones son deterministas y solo se comparte aquella información que cambia, es decir, que no es determinista. Con ello necesitamos controles de consistencia y mecanismos de recuperación si algo falla

Describe brevemente las técnicas de compensación por Predicción y por Extrapolación. ¿En qué se diferencian? (1 punto)

La técnica de compensación por Predicción consiste en ejecutar acciones en el propio cliente nada más recibir el input mientras que la técnica por Extrapolación estima la nueva situación de los datos.
Ambas realizan una convergencia pero en la Extrapolación la predicción que se hace no espera al input. Además, la técnica de Predicción se centra en cómo se verá el cliente mientras que la de Extrapolación se centra en el estado próximo de los objetos basándose en su estado anterior. Es decir, la Predicción consigue la percepción de respuesta inmediata para el jugador y la Extrapolación mantiene el movimiento continuo de los objetos.

¿Qué dos técnicas de compensación se usan en el cliente para reducir la frecuencia de comunicación entre nodos? Enuméralas y explica sus principales diferencias. (1.5 puntos)

Se usan Interpolación en el cliente y Extrapolación en el cliente.
La Interpolación en el cliente utiliza la información de los dos últimos ticks para crear frames intermedios y la Extrapolación hace una estimación de la nueva situación de los datos para después realizar la convergencia con la realidad recibida desde el servidor.
La Interpolación busca conseguir un resultado más fluido y la Extrapolación busca predecir la información en caso de ausencia de datos para mantener la coherencia.

5. Prevención de trampas

En un juego multijugador, la prevención de trampas resulta crucial. Enumera 6 tipos de ataques, y explica brevemente dos de ellos. (2 puntos) .

Algunos de los ataques posibles son: violación de las reglas del juego, exposición de la información, aumento de reflejos, desconexiones, bots para grinding y la creación de múltiples cuentas.
Exposición de la información: ocurre cuando el cliente conoce la información y puede modificarla. Es un problema típico del deterministic lockstep.
Aumento de reflejos: se puede realizar con técnicas como el aimbot o el triggerbot (más difícil de detectar) y se realiza mediante un ejecutable sobre el cliente o packet tampering (modificar los paquetes mientras se envían). Otra forma podría ser generar lag en momentos específicos.
Bots para grinding: se utilizan bots que realizan tareas repetitivas constantemente para obtener beneficios.
Desconexiones: tienen un impacto moderado y se solucionarían desarrollando una lógica para penalizar

Pregunta 2.- Seleccione la respuesta VERDADERA respecto a las vulnerabilidades de un juego multijugador:

a) Hay tres objetivos generales: usuario, cliente y servidor.
b) A través del usuario se puede experimentar vulnerabilidades mediante la colusión.
c) El servidor solamente puede ser comprometido mediante ataques físicos o robos del hardware.
d) Ninguna de las respuestas anteriores es correcta.

¿Qué diferencias existen entre un Map sincronizado y un ConcurrentHashMap en Java? Si se quisiera insertar en el mapa un valor sólo cuando no existe ya la clave, ¿cómo se haría? Sería igual o diferente en cada una de ellas?

¿Qué son las promesas en JavaScript y qué relación tienen con las keywords async/await?

¿Qué herramientas de alto nivel existen en Java para implementar la sincronización condicional? Explica el funcionamiento de 3 herramientas distintas que conozcas

Describe la estrategia de bloqueos a la hora de implementar una base de datos, detallando los distintos tipos y con sus ventajas e inconvenientes.

Describe con tus propias palabras las dos estrategias de bloqueo en una base de datos. (1.5 puntos)

Rellena los espacios con los conceptos que consideres oportunos (0.5 punto).
Para gestionar varias tareas a la vez se usa un ExecutorCompletionService:
• Se pueden _____ tareas (como cualquier executor)
• Dispone del método _____ () para obtener el _____ de las tareas según se van completando.
• Tiene internamente una _____ con los resultados de las tareas creando un esquema _____.
COLA CLIENTE-SERVIDOR ARRAYLIST ENVIAR TAKE FUTURE RECIBIR PRODUCTOR-CONSUMIDOR CALLBACK OFFER

Explica la espera activa y la forma de resolverlo con herramientas de alto nivel

¿Por qué se puede quedar congelada una interfaz de usuario? Pon un ejemplo y describe como se utiliza la programación concurrente para resolver este problema. (1 punto)

Pregunta 4.- Señale cuál de las siguientes afirmaciones es VERDADERA, respecto a los WebSockets:
a) No existe el concepto de petición/respuesta como en API REST.
b) Son un protocolo de comunicación unidireccional
c) No disponen de conexión segura.
d) Ninguna de las respuestas anteriores es correcta.

Pregunta 1.- Se emplea el hilo de despacho de eventos en:
a) Aplicaciones web
b) Aplicaciones de Interfaz de Usuarios
c) Bases de datos
d) Ninguna de las anteriores