



# UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

## DIPLOMADO SUPERIOR EN DESARROLLO DE SOFTWARE EMPRESARIAL

Diplomado que presenta:  
Ana Karen Reyes Monterrosas

### Módulo I **Sistemas Distribuidos**

Docente:  
M. en C. Laura Yadira Dominguez Jalili

Texcoco, Estado de México.

Junio del 2021

# Contenido

## Índice

0.1. Introducción general de los Sistemas Distribuidos . . . . .	3
<b>1. Unidad I. Introducción a la computación distribuida.</b>	<b>3</b>
1.1. Características . . . . .	4
1.2. Tipos de modelos de sistemas . . . . .	5
1.2.1. <b>SISD</b> . . . . .	7
1.2.2. <b>SIMD</b> . . . . .	8
1.2.3. <b>MISD</b> . . . . .	9
1.2.4. <b>MIMD</b> . . . . .	9
1.3. Actividad 1 en clase . . . . .	10
1.4. Sistemas de Memoria Compartida . . . . .	11
1.5. Sistemas de Memoria Distribuida. . . . .	11
1.5.1. Sistemas con memoria compartida . . . . .	11
1.5.2. Sistemas con memoria distribuida . . . . .	12
1.6. Taxonomía de los sistemas distribuidos . . . . .	12
<b>2. Redes.</b>	<b>13</b>
2.1. Actividad 2 en clase. . . . .	13
2.2. Principales componentes de una red . . . . .	13
2.3. Modos de operación y commutación . . . . .	14
2.4. Conmutación . . . . .	14
2.5. Topología de redes . . . . .	15
2.6. El procesamiento paralelo . . . . .	16
2.7. Confiabilidad . . . . .	16
2.8. Velocidad . . . . .	17
2.9. Ejemplo real, alusión a paralelismo . . . . .	17
2.10. Configuraciones Típicas de multiprocesamiento . . . . .	18
2.11. Sincronización . . . . .	19
2.12. Actividad 3 en clase. . . . .	22
<b>3. Modelo OSI</b>	<b>23</b>
3.1. Capa 1 <b>Física</b> . . . . .	23
3.2. Capa 2 <b>Enlace de datos</b> . . . . .	23
3.3. Capa 3 <b>Red</b> . . . . .	23
3.4. Capa 4 <b>Transporte</b> . . . . .	23
3.5. Capa 5. <b>Sesión</b> . . . . .	23
3.6. Capa 6. <b>Presentación</b> . . . . .	24
3.7. Capa 7. <b>Aplicación</b> . . . . .	24
<b>4. Modelo TCP / IP</b>	<b>25</b>
4.1. Actividad 4 en clase. . . . .	26
4.2. Protocolos y paquetes . . . . .	26
4.3. Paradigmas de cómputo y cómo se integran en los modelos arquitectónicos de los sistemas distribuidos. . . . .	26

4.3.1. <b>Modelo Cliente - Servidor</b> . . . . .	26
4.3.2. <b>Peep-to-peer</b> . . . . .	26
4.3.3. <b>Grid</b> . . . . .	28
4.3.4. <b>Proxy</b> . . . . .	29
4.3.5. <b>Clúster</b> . . . . .	30
4.3.6. <b>Applets</b> . . . . .	30
4.4. Arquitectura de capas . . . . .	31
4.5. Sistema distribuido se constituye por: . . . . .	32
4.6. Middleware . . . . .	32
4.7. Corba . . . . .	33
<b>5. Clases prácticas</b>	<b>35</b>
5.1. Actividad 5 en clase . . . . .	35
5.2. Actividad 6 en clase . . . . .	40
5.3. Actividad 7 en clase . . . . .	47
5.4. Actividad 8 en clase . . . . .	49
5.5. Actividad 9 (Tarea) . . . . .	55
5.6. Actividad 10 en clase . . . . .	58
5.7. Actividad 11 en clase . . . . .	62

## 0.1. Introducción general de los Sistemas Distribuidos

En años recientes, el avance en las tecnologías de cómputo y las telecomunicaciones han permitido una gran expansión de los sistemas de información, así como su alta disponibilidad, independientemente de su campo de aplicación.

Las telecomunicaciones permiten la conectividad de un gran número de usuarios ubicados en cualquier parte del mundo por medio de la transmisión de voz, datos o video a través de una gran variedad de dispositivos. Diferentes redes de comunicación de área local (LAN), metropolitanas (MAN), así como de área amplia (WAN), pueden ser accedidas a través de Internet. Esto ha permitido que paralelamente surjan instalaciones de cómputo donde pueden ser desplegadas aplicaciones para realizar procesamiento distribuido de tareas. Estas nuevas facilidades ofrecen a los usuarios y organizaciones una gran flexibilidad para estructurar sus propios sistemas de información de una manera eficiente, así como la oportunidad de interactuar con otros sistemas de información de una manera distribuida. Como consecuencia, esto ha generado una gran dependencia de estos sistemas distribuidos para poder transmitir o procesar información.

Tanenbaum define un sistema distribuido como una colección de computadoras independientes que aparecen ante los usuarios del sistema como una única computadora. El advenimiento de los sistemas distribuidos ha estado soportado en dos importantes innovaciones tecnológicas.

### 1. Unidad I. Introducción a la computación distribuida.

Sistema Distribuido: Conjunto de computadoras que se interconectan en red para colaborar entre sí y realizar una tarea conjunta como si la realizara un solo computador.



Los sistemas distribuidos se implementan en diversas plataformas de hardware. Sus componentes de Hw y Sw no comparten un espacio de memoria común, debido a ello se deben coordinar mediante el paso de mensajes.



### 1.1. Características

- Transparencia
- Consistencia
- Flexibilidad
- Funcionalidad
- Seguridad
- Confiabilidad
- Repartición carga
- Desempeño
- Escalabilidad

**Transparencia:** Característica para ocultar el funcionamiento del sistema y su construcción como, por ejemplo:

- Trasparencia de localización
- Transparencia de migración
- Transparencia de réplica
- Transparencia de concurrencia
- Transparencia de paralelismo
- Transparencia de fallas
- Transparencia de desempeño
- Transparencia de escalabilidad



Tipos de transparencia

1. **Localización:** Ocultar la localización de los datos
2. **Migración:** Los recursos se pueden mover sin cambiar su nombre
3. **Replica:** Ocultar el número de copias existentes
4. **Concurrencia:** Varios usuarios pueden compartir recursos de manera automática
5. **Paralelismo:** Paralelismo sin que el usuario lo perciba
6. **Fallas:** Cuando una computadora del sistema falla, esta es imperceptible para el usuario
7. **Desempeño:** El funcionamiento y velocidad de las máquinas
8. **Escalabilidad:** El usuario ignora cuándo en el sistema agrega otra computadora

**Mantenimiento de consistencia:** Revisión del funcionamiento con el sistema operativo dentro de un sistema distribuido, algunas cosas a considerar son:

- Cache
- Fallas
- Replicación
- Interfaz de usuario
- Reloj

**Flexibilidad:** Facilita modificaciones al diseño inicial.

**Funcionalidad:** El sistema distribuido debe funcionar bajo su régimen y debe ser más eficiente que un sistema centralizado...

**Confiabilidad:** En caso de que una computadora falle, otra la pueda sustituir en la realización de sus tareas asignadas.

**Repartición de la carga:** Análisis de los equipos del sistema y sus diferentes recursos de cómputo.

**Desempeño:** Tiempos de respuesta de una aplicación.

**Escalabilidad:** Permite que a la arquitectura actual se le pueda adicionar más poder de cómputo.

## 1.2. Tipos de modelos de sistemas

- **Servidores estación de trabajo :** Estaciones de trabajo dispersas en un edificio o campus, conectadas entre sí por una red LAN de alta velocidad. Estaciones de trabajo pueden tener discos locales o estaciones sin disco. Las estaciones cuentan con un servidor de archivos.

Uso del disco	Ventajas	Desventajas
Sin disco	Bajo costo, fácil mantenimiento del Hw. Y SW. Simetría y flexibilidad	Gran uso de la red, los servidores de archivos se pueden convertir en cuellos de botella
Paginación, archivos de tipo borrador	Reduce la carga de la red comparado con el caso sin discos	Un costo alto debido al gran número de discos necesarios
Paginación, archivos de tipo borrador, binarios	Reduce todavía más la carga sobre la red	Altos costo, complejidad adicional para actualizar los binarios
Paginación, archivos de tipo borrador, ocultamiento de archivos	Una carga aún menor en la red, también se reduce la carga en los servidores de archivos	Alto costo, problema de consistencia del caché
Sistema local de archivos completo	Escasa carga en la red, elimina la necesidad de los servidores de archivos	Perdida de transparencia

- **Pila de procesadores:** Consiste en un conjunto de procesadores, los cuales se pueden asignar dinámicamente a los usuarios según la demanda.

Los usuarios no disponen de estaciones de trabajo personales, en este modelo se les dan terminales gráficas de alto rendimiento, no se tiene la propiedad de los procesadores debido a que pertenecen a todos compartida mente y un servidor de archivos.



- **Multiprocesadores con memoria compartida y con memoria distribuida:** Taxonomía es la clasificación de las distintas arquitecturas de computadoras.

Clasificación de Flynn:

- SISD
- SIMD
- MISD
- MIMD

La taxonomía de Flynn es la clásica clasificación usada en computación paralela, la cual usa ideas familiares al campo de la computación convencional para proponer una taxonomía de arquitecturas de computadores.

### 1.2.1. SISD

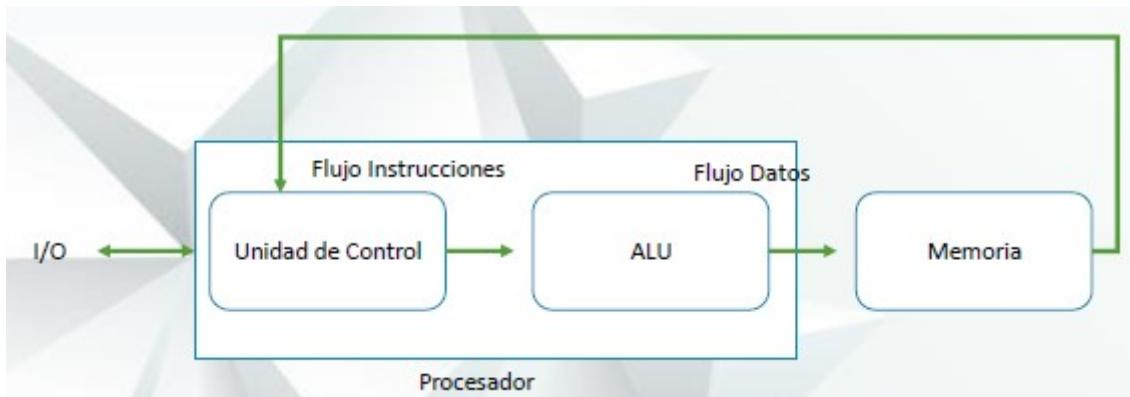
Se refiere a las computadoras convencionales de Von Neumann. Son equipos con un solo procesador que trabaja sobre un solo dato a la vez. A estos equipos se les llama también computadoras secuenciales.

Todas las computadoras SISD utilizan un registro simple llamado: el contador del programa, el cual lleva el conteo de la ejecución serial de las instrucciones.

#### Características SISD

- Flujo único de instrucciones.
- Flujo único de datos.
- Corresponde al modelo estructural básico, con un procesador de instrucciones y un procesador de datos.
- Tiene una única vía de acceso a la memoria principal.
- Este es el modelo tradicional de computación secuencial donde una unidad de procesamiento recibe una sola secuencia de instrucciones que operan en una secuencia de datos.

#### Diagrama SISD



#### Ventajas y desventajas de SISD

##### Ventajas:

- Requiere menos energía.
- No se tiene un protocolo de comunicación complejo.
- Los datos en cuestión se almacenan en una única memoria en la cual se usan técnicas como la segmentación para evitar errores de fragmentación interna.

### Desventajas:

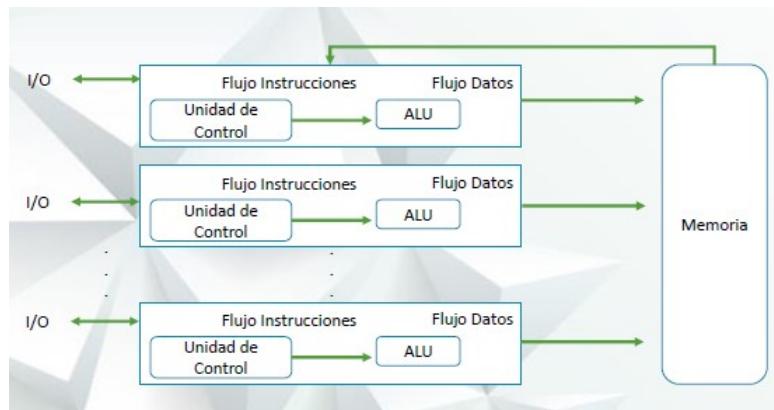
- Velocidad limitada.
- No es útil para aplicaciones amplias.

### 1.2.2. SIMD

Este tipo de sistemas tienen múltiples flujos de datos entrantes y un número de unidades de procesamiento que pueden actuar sobre una sola instrucción en cualquier momento.

Su funcionamiento es el siguiente: un simple controlador envía las instrucciones, una a una, a un arreglo de procesadores que operan en el esquema maestro esclavo. Es decir, las instrucciones son difundidas desde la memoria a un conjunto de procesadores. Así, cada procesador es simplemente una unidad aritmética lógica y se tiene una sola unidad de control. Todos los procesadores ejecutan cada operación recibida al mismo tiempo, por lo cual, cada uno ejecuta la misma instrucción sobre diferentes datos.

Diagrama SIMD



### Ventajas y desventajas de SIMD

#### Ventajas:

- La misma operación en varios elementos se puede realizar usando una sola instrucción.
- El rendimiento del sistema se puede incrementar aumentando el número de núcleos del procesador.
- Mayor velocidad de ejecución que SISD.
- Útil para distintas áreas como Imágenes.

#### Desventajas:

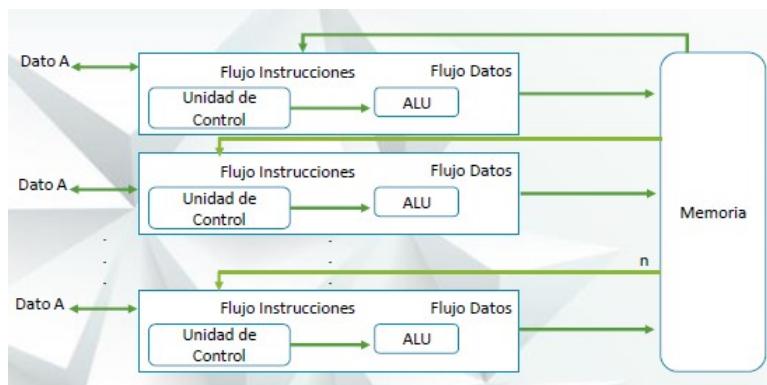
- Existe una comunicación compleja entre el número de núcleos del procesador.
- El costo es más alto que la arquitectura SISD.

### 1.2.3. MISD

Los sistemas con flujo MISD tienen varias unidades de procesamiento que realizan diferentes operaciones mediante la ejecución de diferentes instrucciones en el mismo conjunto de datos. Implementaciones de esta arquitectura no existen realmente.

La idea es descomponer las unidades de procesamiento en fases, en donde cada una se encarga de una parte de las operaciones a realizar. De esta manera, parte de los datos pueden ser procesados en la fase 1 mientras otros son procesados en la 2 otros en la tres, y así sucesivamente. El flujo de información es continuo y la velocidad de procesamiento crece con las etapas.

Diagrama MISD



### 1.2.4. MIMD

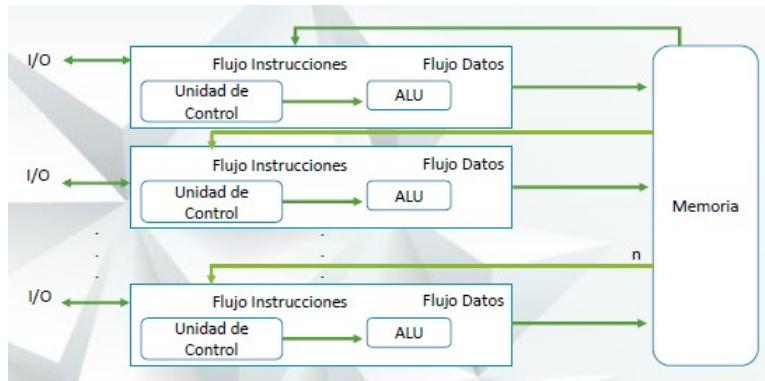
En el sistema que usa la arquitectura MIMD, cada procesador en un sistema multiprocesador puede ejecutar diferentes conjuntos de instrucciones de forma independiente en los diferentes conjuntos de datos en paralelo.

Las maquinas MIMD tienen más de un procesador y cada uno puede ejecutar un programa diferente, con múltiples datos. En muchos sistemas cada procesador tiene acceso a una memoria global, para reducir el tiempo de comunicación entre los procesadores. Además, cada procesador tiene una memoria privada.

Los sistemas MIMD se pueden clasificar en:

- Sistemas de Memoria Compartida.
- Sistemas de Memoria Distribuida.
- Sistemas de Memoria Compartida Distribuida.

Diagrama MIMD



### 1.3. Actividad 1 en clase

Desarrolla un código por cada clasificación de Flynn, en el lenguaje de tu preferencia para explicar los multiprocesadores.

SISD	SIMD	MISD	MIMD
<pre>int x = 3; System.out.println(x); ;</pre>	<pre>int [] arr = {1,2,3,4,5}; for(int i : arr){     System.out.println(arr[i]); }</pre>	<pre>String s = "hola" s = s.substring(0,1); s= s.toUpperCase(); s = "El primer caracter es: " + s; System.out.println(x) ;</pre>	<pre>String[] arrs = {"uno","dos","tres"} for(String s: arrs){     s = s.substring(0,1);     s= s.toUpperCase();     s = "El primer caracter es: " + s;     System.out.println(arr[i]) } ;</pre>

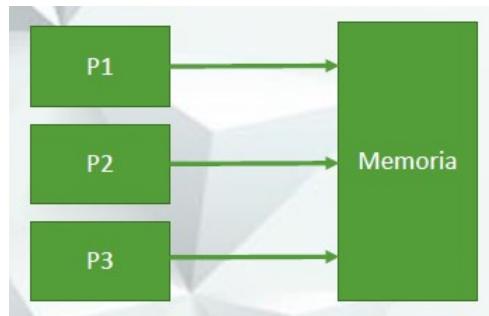
### Tipos de memoria en las arquitecturas SIMD y MIMD

- Sistemas de Memoria Compartida.
- Sistemas de Memoria Distribuida.
- Sistemas de Memoria Compartida Distribuida.

Una clasificación en la cual combinamos los modos de funcionamiento paralelos más comunes de la taxonomía de Flynn (SIMD y MIMD), con los tipos de organización de la memoria (Memoria Compartida (SM) y Memoria Distribuida (DM)).

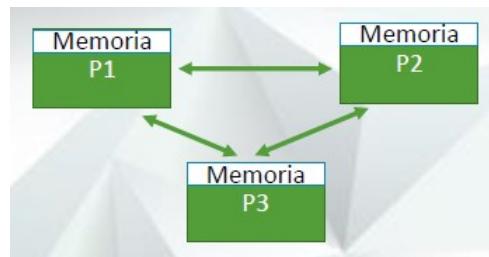
## 1.4. Sistemas de Memoria Compartida

En este tipo de sistemas cada procesador tiene acceso a toda la memoria, es decir hay un espacio de direccionamiento compartido. Las computadoras MIMD con memoria compartida son sistemas conocidos como de multiprocesamiento simétrico (SPM) donde múltiples procesadores comparten un mismo sistema operativo y memoria.



## 1.5. Sistemas de Memoria Distribuida.

Estos sistemas tienen su propia memoria local. Los procesadores pueden compartir información solamente enviando mensajes. Las computadoras MIMD de memoria distribuida son conocidas como sistemas de procesamiento en paralelo masivo (MPP) donde múltiples procesadores trabajan en diferentes partes de un programa, usando su propio sistema operativo y memoria.

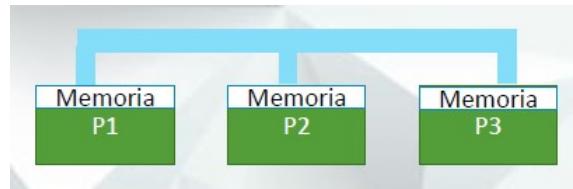


Sistemas de Memoria Compartida Distribuida.

Es una partición de procesadores que tienen acceso a una memoria compartida común, pero sin un canal compartido. Esto es, físicamente cada procesador posee su memoria local y se interconecta con otros procesadores por medio de un dispositivo de alta velocidad, y todos ven las memorias de cada uno como un espacio de direcciones globales.

### 1.5.1. Sistemas con memoria compartida

Soporte de Software:



- Resuelven problemas de secciones críticas.
- Primitivas de sincronización: semáforos, contadores, secuenciadores y monitores.
- Comunicación por espacios de memoria compartida.

Soporte de Hardware:

- Limitado en la escalabilidad.
- Difícil de construir.
- El acceso a la memoria es un cuello de botella.

### **1.5.2. Sistemas con memoria distribuida**

Soporte de Software:

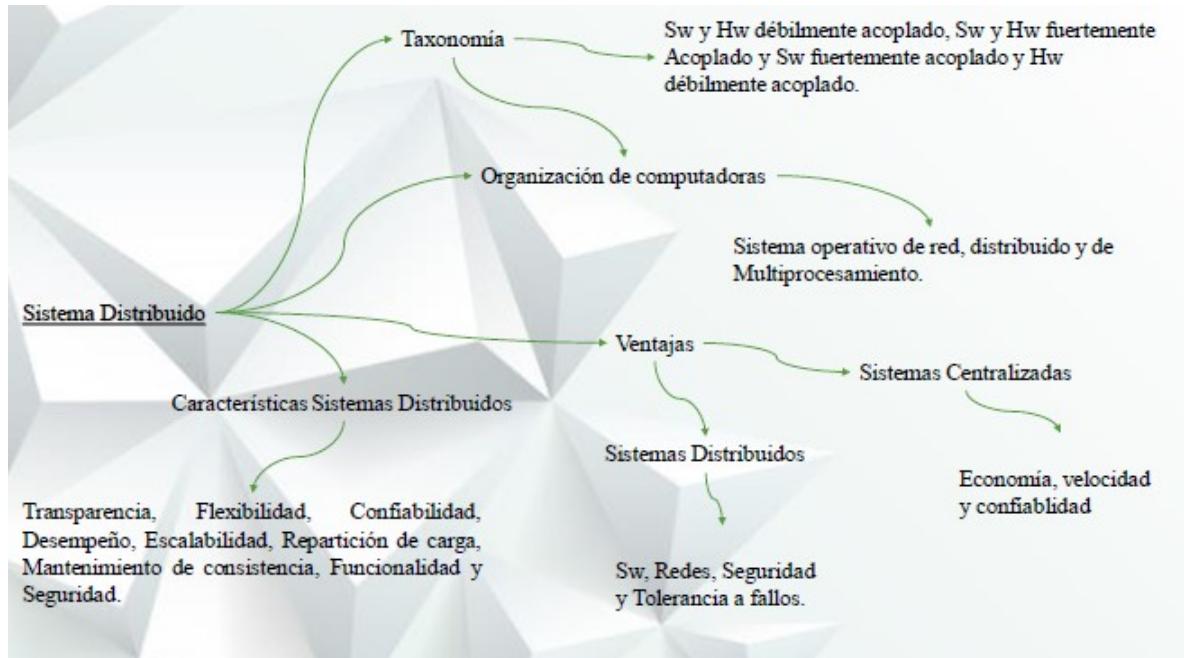
- Pase de mensajes: Trae complicaciones adicionales como perdida de mensajes, perdida de orden, etc.
- Protocolos de comunicación: Buffering y bloqueos

Soporte de Hardware:

- Fácil de construir
- La escalabilidad no está limitada.

### **1.6. Taxonomía de los sistemas distribuidos**

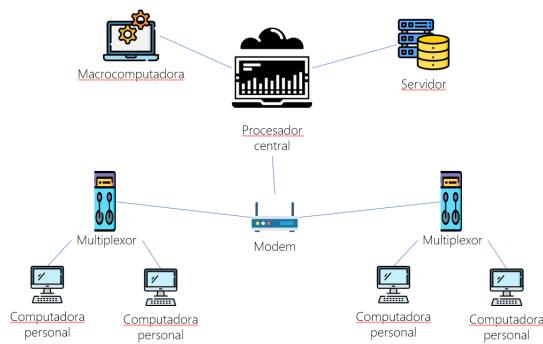
- Sistemas con software débilmente acoplado en hardware débilmente acoplado NFS (Network File System).
- Sistemas con software fuertemente acoplado en hardware fuertemente acoplado; Sistemas operativos de multiprocesador (paralelos).
- Sistemas con software fuertemente acoplado en hardware débilmente acoplado; Sistemas realmente distribuidos (imagen de sistema único).



## 2. Redes.

### 2.1. Actividad 2 en clase.

Desarrolla una topología de red, explicando cada uno de sus componentes visto en el ejemplo del diagrama.



### 2.2. Principales componentes de una red

- **Medios de comunicación:** Transportar datos (líneas telefónicas, canal LAN).
- **Macrocomputadoras:** Computadoras donde están las grandes bases de datos en un ambiente centralizado.

- **Terminales de cómputo:** Dispositivos de I/O.
- **Enrutadores:** Dispositivo que examina las direcciones de la red dentro de un protocolo y encamina paquetes por la ruta.
- **Modem:** Convertir datos digitales seriales a una señal analógica para un canal, y a la Inversa.
- **Multiplexores:** Permite que las terminales compartan la línea de comunicación y reduce el número de líneas usadas.
- Estaciones de trabajo
- Conmutadores
- **Procesador Central:** Maneja el procesamiento de comunicación entre macrocomputadora y los canales de comunicación.
- **Servidores:** Su propósito es proporcionar rendimiento y servicios a estaciones de trabajo.

### 2.3. Modos de operación y conmutación

Modos para trasmisir datos por un enlace de comunicación:

- Comunicación simplex: Cuando los datos viajan en una sola dirección.
- Comunicación half duplex: Permite que los datos viajen en dos direcciones, una a la vez.
- Comunicación full duplex: Los datos viajan simultáneamente en ambas direcciones.

### 2.4. Conmutación

Las redes utilizan comunicación conmutada para transferir datos, esto permite que los dispositivos compartir líneas físicas de comunicación.

1. **Conmutación de circuitos:** Se crea una ruta única e interrumpida entre dos dispositivos que se requieran comunicar, esta ruta no se puede ocupar hasta que se finalice la comunicación.

2. **Conmutación de mensajes:** No se tiene un establecimiento de la ruta, los bloques de mensajes se almacenan en la central de conmutación y se envían uno a la vez. Los bloques no tienen límite de tamaño.

3. **Conmutación de paquetes:** División de datos en fragmentos llamados paquetes que viajan por múltiples rutas entre distintas computadoras. Los paquetes pueden viajar en ambas direcciones, siempre requieren una dirección destino. En la conmutación no se reserva ancho de banda, se toma conforme se necesite, útil en tráfico interactivo.

4. **Conmutación híbrida:** Son variantes que existen en la conmutación de circuitos y paquetes, como conmutación por conexión rápida y conmutación por división de tiempo.

## 2.5. Topología de redes

Referencia al arreglo geométrico que tendrán las conexiones entre las computadoras

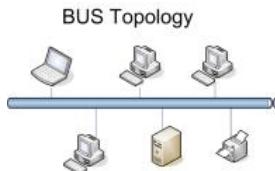
- a) **Topología estrella:** Todas las computadoras se conectan a una computadora central, esta topología no permite la comunicación directa entre dos computadoras que no sean la central.



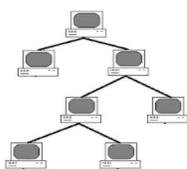
- b) **Topología en anillo:** La red forma un anillo continuo en el cual puede viajar la información.



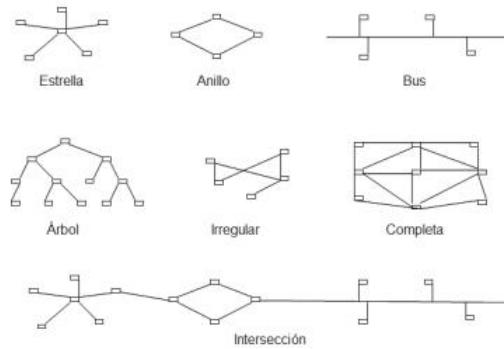
- c) **Topología en bus:** Emplea un solo medio llamado bus, al cual todas las computadoras se conectan.



- d) **Topología en árbol:** Existe una computadora principal que sirve como raíz y única salida externa.



- e) **Topología irregular:** No se respeta un modelo de conexión.
- f) **Topología de intersección:** Es esta topología existe la conexión de dos o más tipos de topologías.
- g) **Topología completa:** Se dan todos los tipos de topologías.



## 2.6. El procesamiento paralelo

Sera una forma de multiprocesamiento, es una situación en que dos o más procesadores ejecutan instrucciones de manera simultánea. En sistemas multiprocesamiento, el administrador del procesador debe coordinar la actividad de cada procesador, así como sincronizar la interacción cooperativa entre las CPU. ¿En que nos ayuda un sistema de procesamiento paralelo? Incremento en la confiabilidad y procesamiento más rápido.

## 2.7. Confiabilidad

Sera la disponibilidad de más de una CPU, si falla un procesador, entonces los otros deben seguir trabajando. Los sistemas paralelos deben diseñarse cuidadosamente de modo que el procesador que fallo pueda informar las fallas y asuman las instrucciones los demás CPU. En un Sistema paralelo influye mucho el SO. Debido a que el realiza la asignación de recursos de forma que los procesadores restantes no se sobrecarguen.

## 2.8. Velocidad

Se logra porque algunas veces las instrucciones pueden procesarse en paralelo, dos o más a la vez. En sistemas muy grandes asignan una CPU para un solo programa o trabajo. Otros asignan una CPU a cada conjunto de trabajo o partes de éste. Algunos subdividen las instrucciones individuales de modo que cada subdivisión pueda procesarse de manera simultánea (y tendríamos programación concurrente).

## 2.9. Ejemplo real, alusión a paralelismo

Un restaurante de comida rápida, tiene un sistema multiprocesamiento sincronizado.

- Procesador 1 (El que recibe la orden) acepta la solicitud, verifica errores y pasa la solicitud.
- Procesador 2 (El empacador) busca la información requerida (i.e. hamburguesa) toma acción a lo que necesita para empacar una hamburguesa.
- Procesador 3 (El cocinero) recupera la información de solicitud y toma acción de cocinar la hamburguesa.
- Procesador 3 envía al procesador 2 y el procesador 2 realiza las instrucciones de empacar.
- Procesador 2 envía a procesador 4 (El cajero).
- Procesador 4 envía respuesta del proceso (la orden).

Y viene la validación de su sistema, lo que ustedes programan. El multiprocesamiento va a tener lugar en diferentes niveles, cada nivel se ejerce con una frecuencia de sincronización diferente.

Nivel Paralelismo	Procesos	Sincronización
Trabajo	Cada trabajo tiene su propio procesador y todos los procesos e hilos son ejecutados en el mismo procesador.	No se requiere sincronización explícita
Proceso	Procesos no relacionados, sin importar el trabajo son asignados a cualquier procesador disponible	Sincronización moderada para rastrear procesos
Hilo	Los hilos se asignan a procesadores disponibles.	Alto grado de sincronización, y instrucciones explícitas del programador.

## 2.10. Configuraciones Típicas de multiprocesamiento

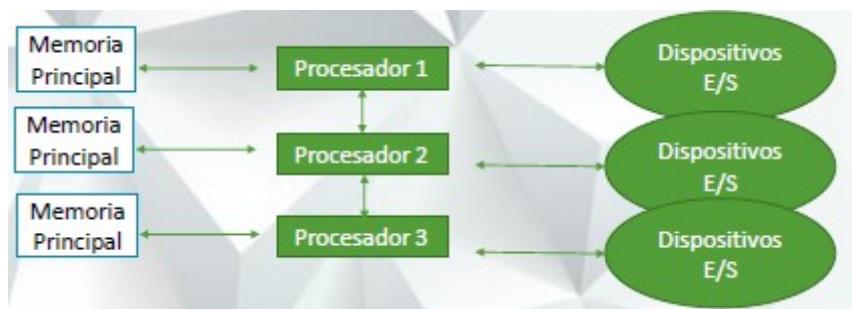
1. Maestro/esclavo.
  2. Débilmente acoplada.
  3. Simétrica.
1. Es un sistema de multiprocesamiento asimétrico, como si fuera un procesador principal en el sistema. Y tienen procesadores esclavos adicionales. Cada procesador esclavo es comandado por el procesador principal. Responsable de todos los archivos, dispositivos, memoria y procesadores.



Los procesadores esclavos pueden acceder directamente a la memoria principal, pero deben enviar todas las solicitudes de E/S a través del procesador maestro.

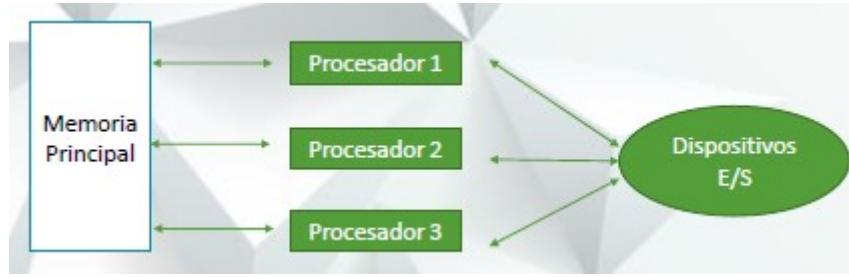
2. Configuración débilmente acoplada.

Cada uno con su propia memoria, dispositivos de E/S, CPU y S.O. Esta configuración se denomina débilmente acoplada porque cada procesador controla sus propios recursos: sus propios archivos. Accesos a la memoria y sus propios dispositivos de E/S, lo cual significa que cada procesador mantiene sus propios comandos y tablas de administración de E/S. La única diferencia entre un sistema multiprocesamiento y débilmente acoplado y una colección de Sistemas multiprocesamiento independiente es que cada procesador puede comunicarse y cooperar con los otros.



3. Configuración simétrica.

Fuertemente acoplada, es la configuración más difícil de implementar porque los procesos deben estar bien sincronizados para evitar bloqueos. En comparación con una configuración débilmente acoplada, es más confiable, usa los recursos de manera efectiva, puede equilibrar las cargas.



## 2.11. Sincronización

1. Prueba e inicio.
2. WAIT Y SIGNAL.
3. Semáforos.

**Prueba e inicio:** : Es una instrucción de máquina única conocida como TS, introducida por IBM para computadoras de un sistema multiprocesamiento 360/370.

Un bit de almacenamiento con cero (libre) o un uno (ocupado), como un subprograma con parámetro de entrada (ubicación de almacenamiento) y retorna un valor (ocupado/libre).

Proceso 1 revisa TS, si está libre entra, de lo contrario espera, es favorable para pocos procesos, pueden llegar varios procesos y esperar esto podría ocasionar inanición (espera indefinida).

**WAIT Y SIGNAL:** WAIT se activa cuando el proceso se encuentra en condición ocupado, WAIT establece un bloque de control de proceso (PCB) en el estado bloqueado y lo vincula con la cola de los procesos que están esperando entrar.

SIGNAL se activa para revisar a TS si está libre u ocupado, verifica la cola de procesos que están esperando entrar para ser ejecutados, y escoge uno estableciéndolo en estado LISTO.

**Semáforos:** Un semáforo es una variable entera no negativa, se usa como una señal binaria (bandera). Un semáforo indica cuando está libre un recurso y el proceso puede utilizarlo S= 0 ocupado.

Num. Edo	Acciones	Operación	Ejecución	Bloqueados	Valor S
0		libre			1
1	P1	ocupado	P1		0
2	P1	libre			1
3	P2	ocupado	P2		0
4	P3	ocupado	P2	P3	0
5	P4	ocupado	P2	P3, P4	0
6	P2	ocupado	P3	P4	0
7		ocupado	P3	P4	0
8	P3	ocupado	P4		0
9	P4	libre			1

**Paralelismo explícito:** Plantear explícitamente cuales instrucciones pueden ejecutarse en paralelo.

**Paralelismo implícito:** Detección automática por el compilador de instrucciones que pueden realizarse en paralelo.

$$\text{Ejemplo1: } A=3*B*C+4/(D+E)*(F-G)$$

### Secuencial

Paso	Operación	Resultado
1	(F-G)	T1
2	(D+E)	T2
3	T2*T1	T1
4	4/(T1)	T2
5	3*B	T1
6	T1*C	T1
7	T1+T2	A

$$B=2$$

$$C=2$$

$$D=2$$

$$E=2$$

$$F=10$$

$$G=5$$

$$\text{Res}=(D+E)*(F-G)$$

$$\text{Res1}=3*B$$

$$\text{Res2}=\text{Res1}*C$$

$$\text{Res3}=\text{Res2}+4$$

$$A=\text{Res3}/\text{Res}$$

$$\text{Eejmplo 2: } A=3*B*C+4 / (D+E)*( F-G)$$

$$A=P1*C+4/P2*P3$$

$$A=P1+4/P2$$

$$A=P1+P1$$

## Paralelismo

Paso	Procesador	Operación	Resultado
1	1	$3*B$	T1
	2	$(D+E)$	T2
	3	$(F-G)$	T3
2	1	$T1*C$	T4
	2	$T2*T3$	T5
3	1	$4/T5$	T1
4	1	$T4+T1$	A

For paralelo:

```
for(i=1; i<=3; i++)
    a(i)=b(i)+c(i)
```

P1	P2	P3
$a(1)=b(1)+c(1)$	$a(2)=b(2)+c(2)$	$a(3)=b(3)+c(3)$

While paralelo

```
i=0;
while(i<3)
    a(i)=i+1
    i+=1
```

P1	P2	P3
$a(0)=(0)+1$	$a(1)=(1)+1$	$a(2)=(2)+1$

## 2.12. Actividad 3 en clase.

Realiza los siguientes ejercicios usando paralelismo.

Primer ejercicio: multiplicación de matrices con un total de procesadores = 3.

K	L		*	A	B	C
M	N			D	E	F
O	P					

Solución del primer ejercicio.

Paso	Procesador	Opeiaqrón	Rasultedo
1	1	K*A	T1
	2	L*D	T2
	3	T1+T2	A
2	1	M*A	T1
	2	N*D	T2
	3	T1+T2	B
3	1	O*A	T1
	2	P*D	T2
	3	T1+T2	C
4	1	K*B	T1
	2	L*E	T2
	3	T1+T2	D
5	1	M*B	T1
	2	N*E	T2
	3	T1+T2	E
6	1	O*B	T1
	2	P*E	T2
	3	T1+T2	F
7	1	K*C	T1
	2	L*F	T2
	3	T1+T2	G
8	1	M*C	T1
	2	N*F	T2
	3	T1+T2	H
9	1	O*C	T1
	2	P*F	T2
	3	T1+T2	I

Matriz Resultante

A	D	G
B	E	H
C	F	I

Segundo ejercicio:  $Y=A+B*C+D$

Solución del segundo ejercicio.

Paso	Procesador	Operación	Resultado
1	1	$B*C$	T1
2	1	$A+T1$	T2
3	1	$T2+D$	Y

### 3. Modelo OSI

Open System Interconnection. Utiliza 7 capas para organizar una red en módulos funcionales bien definidos.

- Una capa se creará en situaciones en las que se necesite un nivel diferente de abstracción.
- Cada capa deberá efectuar una función bien definida.
- La función que realizará cada capa deberá seleccionarse con la intención de definir protocolos normalizados internacionalmente.
- Los límites de las capas deberán seleccionarse tomando en cuenta la minimización el flujo de información a través de las interfaces.

#### 3.1. Capa 1 Física

Define las especificaciones eléctricas y funcionales. Características como niveles de voltaje, temporización de cambios de voltaje, velocidad de datos, distancias de Transmisión, conectores físicos, etc.

#### 3.2. Capa 2 Enlace de datos

Direccionamiento físico MAC, la topología de red, acceso a la red, notificación de errores, entrega ordenada de tramas y control de flujo de datos (Switch).

#### 3.3. Capa 3 Red

Selección de ruta entre dos sistemas de hosts, direccionamiento y enrutamiento (Router).

#### 3.4. Capa 4 Transporte

Responsable de la confiabilidad de transporte entre dos hosts Utiliza dispositivos de detección y recuperación de errores, asignación de protocolos como TCP y UDP.

#### 3.5. Capa 5. Sesión

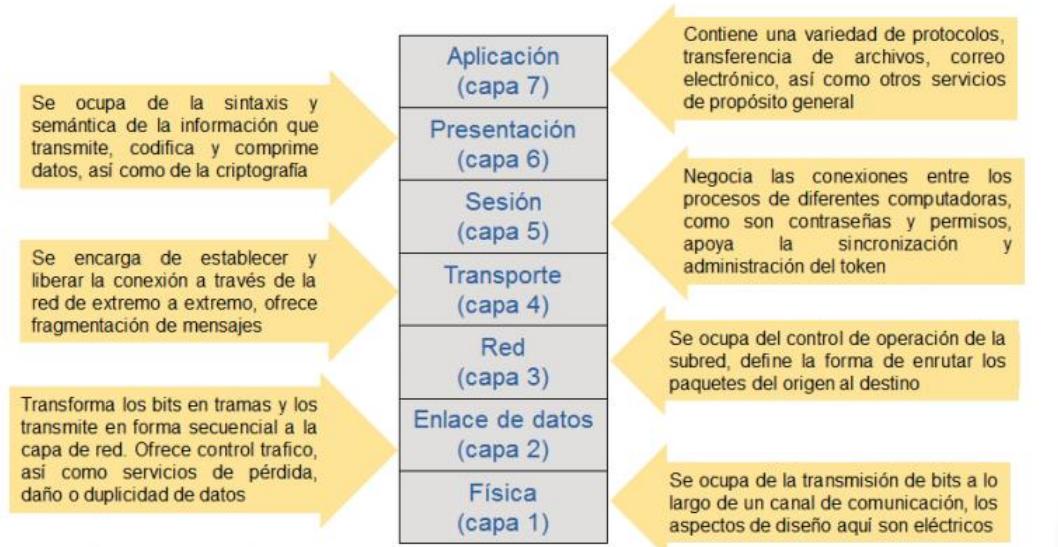
Permite indicar puntos en la comunicación de los hosts para recuperar o continuar en uno de estos puntos. Mantiene un control de dialogo, puntos de sincronización y permite recuperar los envíos hasta un punto anterior.

### 3.6. Capa 6. Presentación

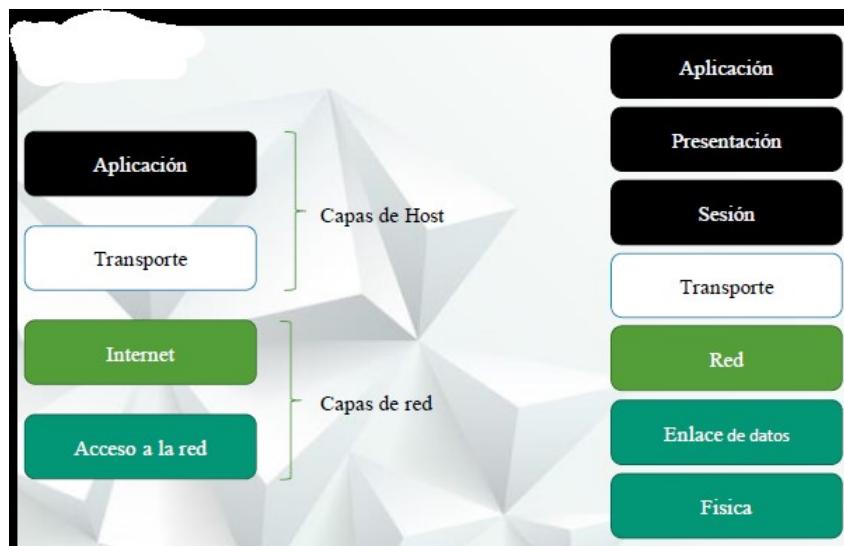
Define los formatos de representación de los datos como caracteres, números o imágenes. Realiza conversión de formatos, cifrado y compresión.

### 3.7. Capa 7. Aplicación

Servicios específicos a los usuarios como correo, envío de mensajes, etc. Es la vista principal de las aplicaciones y de acuerdo a lo que se va a realizar se tienen protocolos como es HTTP, SMTP (correo), DNS (sistema nombre de dominio, etc.).



## 4. Modelo TCP / IP



**Aplicación:** Servicio específico a usuarios y de acuerdo a la aplicación se utiliza el protocolo adecuado.

**Transporte:** En la capa internet no se garantiza el envío de paquetes, el orden inadecuado y tamaño pequeño de paquetes. La capa transporte realiza la transmisión de datos óptima de cualquier tamaño y libre de errores el nivel de transporte se va a comprometer a segmentar en paquetes, enviarlos en un orden adecuado, ensamblar en el destino y asegurarnos que no se perdió ninguno y son correctos. Utiliza protocolos TCP y UDP.

**Internet:** Comportamiento de intercambio de paquetes en la red. Solo se tienen un protocolo a usar que es IP (IPv 4 o IPv 6).

**Acceso a la red:** No se define en el modelo, solo indica que debemos disponer de un enlace entre equipos, ya sea wifi, ethernet etc.

#### **4.1. Actividad 4 en clase.**

Realiza un ejemplo de transferencia de datos utilizando como referencia el modelo TCP / IP.

Actividad 1:

Modelo TCP/IP Actual en un correo electrónico.

Aplicación: El usuario escribe y manda un correo electrónico. El servidor procesa el correo electrónico el contenido del correo electrónico y después identifica de dónde viene y a dónde va.

Transporte: Se encarga de llevar el paquete de un host a otro.

Red: Define la forma de llegar a la ruta adecuada.

Enlace: Todo lo que se ha obtenido, se transforma en bits para ser procesado por la capa física.

Física: Se referiría a las especificaciones físicas de la computadora en la que se manda el correo.

#### **4.2. Protocolos y paquetes**

Protocolos son un conjunto de reglas para la interacción de procesos concurrentes en sistemas distribuidos, en diferentes campos como sistemas operativos, redes de computadoras o comunicación de datos. Uno de los más usados es TCP/IP (Transmission Control Protocol / Internet Protocol), este conjunto tiene menor número de capas que el modelo OSI, por lo que se considera un incremento en su eficiencia.

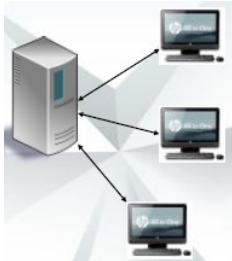
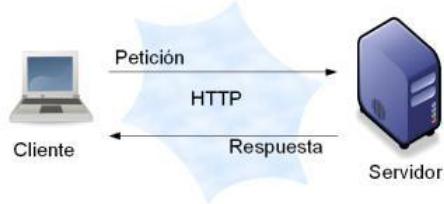
#### **4.3. Paradigmas de cómputo y cómo se integran en los modelos arquitectónicos de los sistemas distribuidos.**

La arquitectura de un sistema es su relación y estructura entre los componentes y como están relacionados. Algunos modelos arquitectónicos principales de los sistemas distribuidos son:

- Paradigmas cliente - servidor
- Pree - to - peer
- Grid
- Proxy
- Cluster

##### **4.3.1. Modelo Cliente - Servidor**

Una de las arquitecturas más citadas, los procesos toman el rol de ser clientes o servidores, los procesos de cliente interactúan con los procesos de servidor individuales en equipos de



trabajo.

Cliente servidor para varios clientes, grupo de servidores interconectados en modelo cliente servidor.

#### 4.3.2. Peep-to-peer

En el modelo cliente servidor tradicional, se tiene dos nodos que son clientes o servidores y los clientes solicitan servicios y el servidor los proporciona. Sin embargo, en los sistemas peer to peer (P2P) no se requiere una infraestructura dedicada, cada peer puede tomar el papel tanto de servidor como de cliente al mismo tiempo.

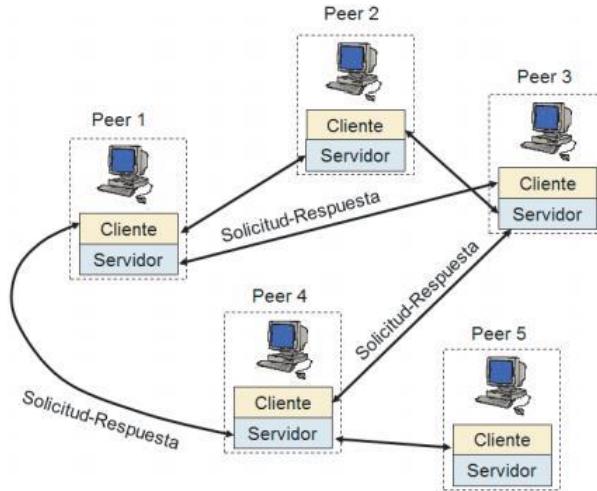
Beneficios:

- Nodos comparten recursos
- Se pueden desplegar algoritmos distribuidos
- Escalamiento más fácil del sistema
- Ahorro de costos
- Flexibilidad
- Ningún punto único de falla
- Mayor robustez del sistema

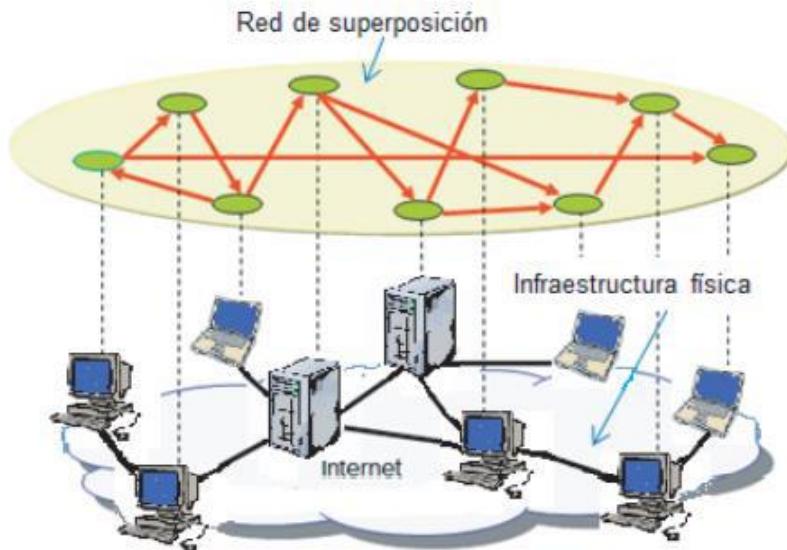
eMule mejora de eDonkey 2000 y con la colaboración de la comunidad Open.

Millones de usuarios en todo el mundo se acostumbraron a compartir y descargar archivos a través de las redes eDonkey y Kad, tras conectarse a una amplia lista de servidores.

Tiene una infraestructura de comunicación par a par, se forma por grupo de nodos ubicados en una red física, estos nodos construyen una abstracción de red conocida como red superpuesta.



Una red superpuesta se establece para cada sistema P2P a través de conexiones TCP o HTTP, la red superpuesta construye túneles lógicos entre pares de nodos para implementar su propio mecanismo de enrutamiento para transportar sus mensajes.



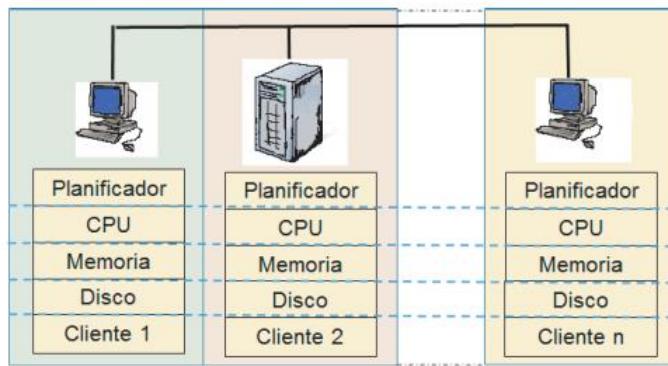
#### 4.3.3. Grid

Infraestructura de gestión de recursos distribuidos que se centra en el acceso coordinado a los recursos informáticos remotos. A diferencia del cómputo de clúster, el computo grid tiende a ser más heterogéneo y disperso geográficamente. Los recursos que son integrados por

una infraestructura grid con plataformas de cómputo dedicadas a super computadoras de alta gama o clúster de propósito general.

#### Beneficios del cómputo Grid

- Explotación de recursos infrautilizados
- Capacidad de CPU paralelos
- Recursos virtuales y organizaciones virtuales
- Acceso a recursos adicionales
- Balanceo de recursos
- Fiabilidad
- Mejora en gestión de infraestructura de TI distribuidos



#### 4.3.4. Proxy

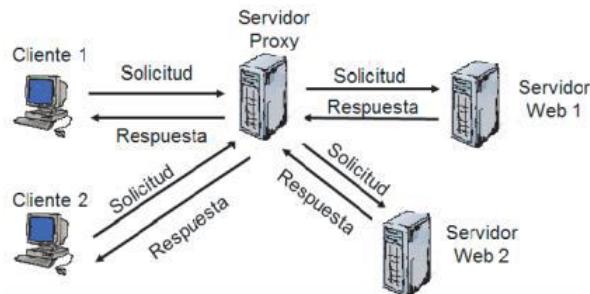
Es un servidor que se emplea como intermediario entre las peticiones de recursos que realiza un cliente a otro servidor.

Computadora A solicita un recurso a la computadora C, realizará la petición a la computadora B, que a su vez trasladará la petición a la computadora C, con lo cual la computadora C no sabrá que la petición fue de la computadora A.

Esto se usa para soportar:

- Proporcionar caché
- Control de acceso
- Registro del tráfico
- Prohibir cierto tipo de tráfico
- Mejorar el rendimiento
- Mantener el anonimato

El proxy más conocido es el servidor proxy web, su función principal es interceptar la navegación de los clientes por páginas web por motivos de seguridad, rendimiento, anonimato, etc.



#### 4.3.5. Clúster

Conjuntos de computadoras construidos mediante el uso de hardware común y se comportan como si fueran una única computadora. Uso de clústeres va desde aplicaciones de supercómputo, servidores web y comercio electrónico hasta software de misiones críticas y bases de datos de alto rendimiento.

Servicios de un clúster:

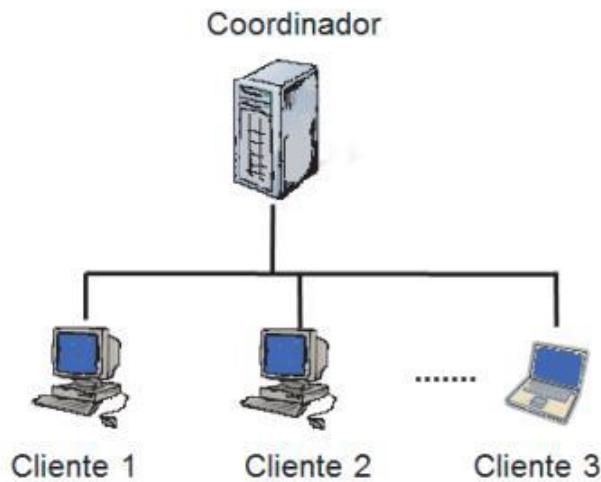
- Alto rendimiento
- Alta disponibilidad
- Escalabilidad
- Balanceo de carga

Tipos de clúster:

- **Clúster homogéneo:** Cuando todas las computadoras tienen la misma configuración en hardware y sistema operativo.
- **Clúster semihomogéneo:** Cuando las computadoras tienen diferente rendimiento, pero guardan una similitud con respecto a su arquitectura y sistema operativo.
- **Clúster heterogéneo:** Cuando las computadoras tienen diferente hardware y sistema operativo.

#### 4.3.6. Applets

Un applet es un código que se ejecuta en el contexto de otro programa, como un navegador Web, el código se descarga en el navegador y se ejecuta allí. Un applet carece de uso independiente y son muy utilizados en aplicaciones móviles, los applets no sufren los retrasos o variabilidad de ancho de banda, aunque un applet tiene privilegios restringidos de seguridad, no tiene acceso al sistema de archivos local. Un applet es no confiable a menos que lleve una



firma digital de una entidad especificada como confiable.

Ejemplos de los applets más comunes son:

- Java applets
- Animaciones Flash
- Windows media player
- Modelos 3D

Los applets son considerados algo en desuso, sin embargo Oracle considera que aunque su uso sea bajo, siempre es interesante ver con ejemplos sus posibilidades técnicas y aprender de ellas.

- Funciones especializadas, como por ejemplo, applets para calcular el valor del ángulo inscrito en una circunferencia y circuncentro de un triángulo.
- Mostrar efectos visuales e imágenes con sonidos y agregar efectos sonoros.
- Gráficos interactivos, reaccionando a acciones que se toman con el mouse sobre el gráfico.
- Crear diagramas y gráficas, como por ejemplo, la clásica gráfica de trozos de pastel.
- Juegos sencillos.

#### 4.4. Arquitectura de capas

Un sistema complejo puede ser dividido en capas, las capas superiores hacen uso de los servicios ofrecidos por las capas inferiores. Para los sistemas distribuidos, los servicios se organizan de una manera vertical como capas de servicio, un servicio puede ser proporcionado por uno o más procesos del servidor y con los procesos de cliente se mantiene una visión de

todo el sistema.

Ejemplo:

Ajustar la hora entre los clientes, puede realizarse por un servicio de hora de red a través de internet, usando el protocolo de tiempo de red (NTP).



#### 4.5. Sistema distribuido se constituye por:

1. Plataforma
2. Middleware
3. Aplicaciones y servicios

**Plataforma:** Se compone de las capas de hardware y software de nivel más bajo, son implementadas de manera independiente en cada equipo, conduciendo a la interfaz de programación del sistema hasta un nivel que facilita la comunicación y la coordinación entre los procesos.

**Middleware:** Es un software que tiene como función proporcionar un modelo de programación conveniente a los programadores de aplicaciones.

Ejemplos de middleware:

- CORBA
- JAVA
- RMI

**Aplicaciones y servicios:** Prestaciones que ofrece el sistema distribuido a los usuarios aplicaciones distribuidas.

#### 4.6. Middleware

Son representados por estándares como ODBC, OLE, DCOM y CORBA, permite distribuir datos y procesos a través de un sistema multitarea, una red local, una red remota o internet, los servicios de middleware se clasifican como:

- Servicios de desarrollo
- Servicios de administración

Un objetivo principal es la transparencia en los sistemas distribuidos por medio de:

- Ofrecer la capacidad, así como solicitar y recibir de manera transparente al sistema.
- Liberar a los diseñadores y administradores del sistema de problemas derivados por la complejidad del sistema operativo.

Middleware es representado por procesos u objetos en un conjunto de equipos que interactúan entre sí para implementar la comunicación de equipos que interactúan entre sí para implementar la comunicación y el intercambio de recursos de soporte para las aplicaciones distribuidas. Para conservar la transparencia se desarrolla diversas versiones de un sistema middleware, donde cada versión sea confeccionada para una clase específica de aplicaciones.



Los middlewares basados en componentes permiten ocultar detalles de bajo nivel asociados al middleware, gestión de las complejidades de establecer aplicaciones distribuidas con propiedades no funcionales apropiadas como la seguridad y el apoyo apropiado de estrategias de implementación.

Los middlewares basados en objetos distribuidos proporcionan un modelo de programación basado en principios orientada a objetos y lleva los beneficios del enfoque. Principales ejemplos de middleware basado en objetos distribuidos son Java RMI y CORBA.

#### **4.7. Corba**

Es una herramienta middleware que facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos tanto en hardware como en software.

Ejemplos:

- Distintos sistemas operativos (Unix, Windows, MacOs, etc.)
- Distintos protocolos de comunicación (TCP/ IPX, IPX, etc.)
- Distintos lenguajes de programación (java, C, C++, etc.)
- Distinto hardware

## 5. Clases prácticas

### 5.1. Actividad 5 en clase

Desarrolla los siguientes programas en clase.

- **Programa 1:** Hilo de nombres con metodo sleep y random.
- **Programa 2:** Hilo con método for y con método yield.
- **Programa 3:** Interfaz y funcionamiento de una calculadora con metodo join y validación de datos.

*Solución del primer programa.*

Clase Hilo1.java

```
public class Hilo1 extends Thread{  
  
    String nombres;  
  
    public Hilo1(String nombre){  
        this.nombres=nombre;  
    }  
  
    @Override  
    public void run(){  
        int x = (int) (Math.random()*3000);  
        try{  
            Thread.sleep(x);  
            System.out.println("Soy: "+nombres+", durmió "+x);  
        }catch(InterruptedException ex){  
            System.out.println("Error hilo: "+ex.getMessage());  
        }  
    }  
}
```

Clase DiplomadosHilos.java (main)

```
public class DiplomadosHilos {  
  
    public static void main(String[] args) {  
        // TODO code application logic here  
        Hilo1 h1 = new Hilo1("Saul");  
        Hilo1 h2 = new Hilo1("Karen");  
        Hilo1 h3 = new Hilo1("Carlos");  
    }  
}
```

```

        h1.start();
        h2.start();
        h3.start();
        System.out.println("Corriendo hilos");
    }

}

```

```

run:
Finalizar el programa
Soy Samuel. Durmio 239
Soy Estela. Durmio 456
Soy Diego. Durmio 984
BUILD SUCCESSFUL (total time: 1 second)

```

Figura 1: Salida del primer programa, Actividad 5

*Solución del segundo programa.*

Clase DiplomadosHilos2.java (main)

```

public class DiplomadosHilos2 {

    public static void main(String[] args) {
        // TODO code application logic here
        ElHilo h1 = new ElHilo("Saul");
        ElHilo h2 = new ElHilo("Naranja");

        h1.start();
        h2.start();
    }

    static class ElHilo extends Thread {
        String nombre;

        public ElHilo(String nombre) {
            this.nombre = nombre;
        }

        public void run() {

```

```

        for (int i = 0; i < 6; i++) {
            System.out.println("nombre es: " + nombre + " / " + i);
            yield();
        }
    }
}

```

```

diplomadohilos2.DiplomadoHilos2 > main > h1 >
Output - DiplomadoHilos2 (run) ×
run:
El nombre es: Carmen - 0
El nombre es: Carmen - 1
El nombre es: Carmen - 2
El nombre es: Carmen - 3
El nombre es: Carmen - 4
El nombre es: Carmen - 5
El nombre es: Raul - 0
El nombre es: Raul - 1
El nombre es: Raul - 2
El nombre es: Raul - 3
El nombre es: Raul - 4
El nombre es: Raul - 5
BUILD SUCCESSFUL (total time: 0 seconds)

```

Figura 2: Salida del segundo programa, Actividad 5

*Solución del tercer programa.*

Clase Multiplicacion.java

```

public class Multiplicacion extends Thread{
    int a,b,r;

    public Multiplicacion(Integer a, Integer b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public void run(){

        r = a*b;
        System.out.println("La multiplicacion es: "+r);
    }
}

```

Clase Resta.java

```
public class Resta extends Thread{
    int a,b,r;

    public Resta(Integer a, Integer b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public void run(){

        r = a-b;
        System.out.println("La resta es: "+r);
    }
}
```

Clase Suma.java

```
public class Suma extends Thread{
    int a,b,r;

    public Suma(Integer a, Integer b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public void run(){

        r = a+b;
        System.out.println("La suma es: "+r);
    }
}
```

Clase Calculadora.java (main)

```
public class Calculadora extends javax.swing.JFrame {
    public Calculadora() {
        initComponents();
    }
    private void salirActionPerformed(java.awt.event.ActionEvent evt) {
```

```

    // TODO add your handling code here:
    System.exit(0);
}

private void iniciarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    int a, b;
    a = Integer.parseInt(valor1.getText());
    b = Integer.parseInt(valor2.getText());
    Suma s = new Suma(a, b);
    Resta r = new Resta(a, b);
    Multiplicacion m = new Multiplicacion(a, b);

    System.out.println("a= " + a);
    System.out.println("b= " + b);

    s.start();
    r.start();
    m.start();

    try {
        s.join();
        r.join();
        m.join();
    } catch (InterruptedException ex) {
        Logger.getLogger(Calculadora.class.getName()).log(Level.SEVERE, null, ex);
    }

    resultadoSuma.setText(String.valueOf(s.r));
    resultadoResta.setText(String.valueOf(r.r));
    resultadoMultiplicacion.setText(String.valueOf(m.r));
}

public boolean validar(String numero) {
    int n;
    try {
        n = Integer.parseInt(numero);
        return true;
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
        return false;
    }
}
}

```



Figura 3: Salida del tercer programa, Actividad 5

## 5.2. Actividad 6 en clase

Desarrolla los siguientes programas en clase.

- **Programa 1:** Formas de instanciar un hilo.
- **Programa 2:** Bola magica.
- **Programa 3:** Grupo Procesos.

*Solución del primer programa.*

Clase hilos.java

```
public class hilos implements Runnable{
    private String mensaje;

    public hilos(String m){
        this.mensaje=m;
    }

    public void run(){
        System.out.println("hola for interno: "+mensaje);
        dormirmensaje();
    }

    private void dormirmensaje(){
        try {

```

```

        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}
}
```

Clase FormasdeInstanciarHilos.java (main)

```

public class FormasdeInstanciarHilos {

    public static void main(String[] args) {
        // TODO code application logic here
        ExecutorService executor = Executors.newFixedThreadPool(5);
        for (int i = 0; i < 10; i++) {
            Runnable h=new hilos("for externo: "+i);
            executor.execute(h);
        }
        executor.shutdown();
        while(!executor.isTerminated()){
        }
        System.out.println("Finaliza hilos");
    }
}
```

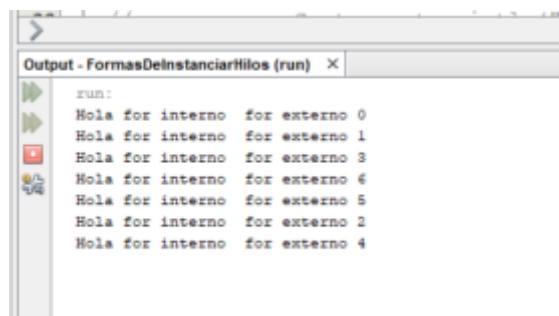


Figura 4: Salida del primer programa, Actividad 6

*Solución del segundo programa.*

Clase FraseAmor.java

```
public class FraseAmor extends Thread{
    String frase;

    public void run(){
        int caso=(int)(Math.random()*5);

        switch(caso){
            case 1: frase="Echale ganas a tu trabajo";
            break;
            case 2: frase="Trabaja duro todos los días";
            break;
            case 3: frase="Nunca renuncies a tus sueños";
            break;
            case 4: frase="Conoce gente nueva en tu trabajo";
            break;
            case 5: frase="Comparte conocimientos";
            break;
            default: frase="Amor...";
        }
    }
}
```

Clase FraseTrabajo.java

```
public class FraseTrabajo extends Thread{
    String frase;

    public void run(){
        int caso=(int)(Math.random()*5);

        switch(caso){
            case 1: frase="Amate a ti mismo";
            break;
            case 2: frase="Ama a los que te rodean";
            break;
            case 3: frase="Ama todo lo que haces";
            break;
            case 4: frase="Vendrán cosas nuevas en tu vida";
            break;
            case 5: frase="Nunca te rindas";
            break;
            default: frase="Trabajo...";
        }
    }
}
```

```
    }  
}  
  
}
```

### Clase frasebolamagica.java (main)

```
public class frasebolamagica extends javax.swing.JFrame {  
    private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {  
        // TODO add your handling code here:  
        System.exit(0);  
    }  
  
    private void btnIniciarActionPerformed(java.awt.event.ActionEvent evt) {  
        // TODO add your handling code here:  
        FraseTrabajo t= new FraseTrabajo();  
        FraseAmor a= new FraseAmor();  
        t.start();  
        a.start();  
        try {  
            t.join();  
            a.join();  
        } catch (InterruptedException ex) {  
            Logger.getLogger(frasebolamagica.class.getName()).log(Level.SEVERE, null, ex);  
        }  
        frase1.setText(t.frase);  
        frase2.setText(a.frase);  
    }  
}
```



Figura 5: Salida del segundo programa, Actividad 6

*Solución del tercer programa.*

Clase Hilo1.java

```
public class Hilo1 extends Thread{
    int limite, Timesleep;

    public Hilo1(int limite, int Timeslepp) {
        this.limite = limite;
        this.Timesleep=Timesleep;
    }

    public void run(){
        for (int i = 0; i < limite; i++) {
            System.out.println("Hilo1: "+i);
            dormir();
        }
    }

    public void dormir(){
        try {
            Hilo1.sleep(Timesleep);
        } catch (InterruptedException ex) {
            System.out.println("Error"+ex.getMessage());
        }
    }
}
```

Clase Hilo2.java

```
public class Hilo2 implements Runnable{
    int limite, Timesleep;

    public Hilo2(int limite, int Timeslepp) {
        this.limite = limite;
        this.Timesleep=Timesleep;
    }

    public void run(){
        for (int i = 0; i < limite; i++) {
            System.out.println("Hilo2: "+i);
            dormir();
        }
    }

    public void dormir(){
```

```

        try {
            Thread.sleep(Timesleep);
        } catch (InterruptedException ex) {
            System.out.println("Error"+ex.getMessage());
        }
    }
}

```

Clase Hilo3.java

```

public class Hilo3 extends Thread {
    int limite, Timesleep;

    public Hilo3(int limite, int Timeslepp) {
        this.limite = limite;
        this.Timesleep=Timesleep;
    }

    public void run(){
        for (int i = 0; i < limite; i++) {
            System.out.println("Hilo3: "+i+5);
            dormir();
        }
    }

    public void dormir(){
        try {
            Hilo3.sleep(Timesleep);
        } catch (InterruptedException ex) {
            System.out.println("Error"+ex.getMessage());
        }
    }
}

```

Clase Prioridades.java (main)

```

public class Prioridades {

    public static void main(String[] args) {
        // TODO code application logic here
        Thread h1=new Thread(new Hilo1(10, 1000));
        Runnable h2=new Hilo2(5,500);
        Hilo3 h3=new Hilo3(6, 500);
    }
}

```

```
        h3.start();
        h1.start();
        h2.run();

        h1.setPriority(10); //Maxima prioridad
        h1.setPriority(1); //Minima prioridad

    }
}
```

The screenshot shows the 'Output' window of an IDE during the execution of a program named 'Prioridades'. The window title is 'Output - Prioridades (run)'. The output text is as follows:

```
run:
Hilo3 5
Hilo1 0
Hilo3 6
Hilo1 1
Hilo2 7
Hilo2 8
Hilo3 9
Hilo3 0
Hilo3 1
Hilo3 2
Hilo3 3
Hilo3 4
Hilo3 5
BUILD SUCCESSFUL (total time: 5 seconds)
```

Figura 6: Salida del tercer programa, Actividad 6

### 5.3. Actividad 7 en clase

Desarrolla un cronometro con hilos, hilo segundos, hilo minutos con sincronización sleep.

*Solución del programa.*

Clase Hilosegundos.java

```
public class Hilosegundos extends Thread {

    @Override
    public void run() {

        for (int i = 0; i < 60; i++) {
            for (int j = 0; j < 60; j++) {
                if (j < 10) {
                    System.out.println(":0" + j);
                } else {
                    System.out.println(j);
                }

                try {
                    Thread.sleep(1000);
                } catch (InterruptedException ex) {
                    System.out.println("Error sleep" + ex.getMessage());
                }
            }
        }
    }
}
```

Clase Hilominutos.java

```
public class Hilominutos extends Thread {

    @Override
    public void run() {
        for (int i = 0; i < 60; i++) {
            for (int j = 0; j < 60; j++) {
                if (i < 10) {
                    System.out.print("0" + i);
                } else {
                    System.out.print(i);
                }

                try {

```

```

        Thread.sleep(1000);
    } catch (InterruptedException ex) {
        System.out.println("Error sleep" + ex.getMessage());
    }
}
}
}
}
```

Clase Cronometro.java (main)

```

public class Cronometro {

    public static void main(String[] args) {
        // TODO code application logic here
        Hilominutos h1=new Hilominutos();
        Hilosegundos h2=new Hilosegundos();

        h1.start();

        try {
            h1.sleep(50);
        } catch (InterruptedException ex) {
            System.out.println("Error sleep"+ ex.getMessage());
        }

        h2.start();
        try {
            h2.sleep(50);
        } catch (InterruptedException ex) {
            System.out.println("Error sleep"+ ex.getMessage());
        }
    }
}
```

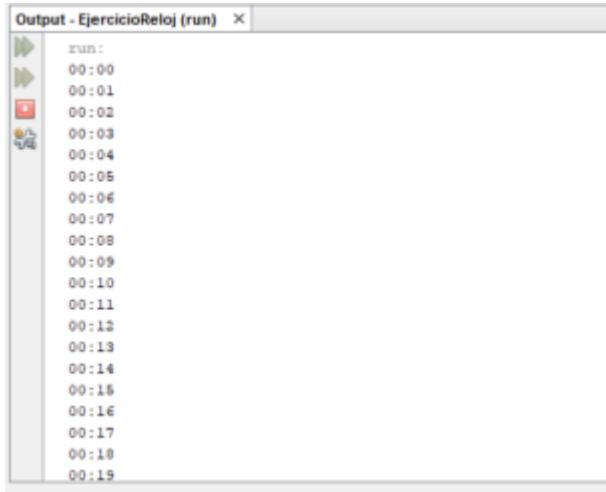


Figura 7: Salida del programa, Actividad 7

#### 5.4. Actividad 8 en clase

Desarrolla los siguientes programas en clase.

- **Programa 1:** GUI cronometro con horas, min, seg. Clase Observable.
- **Programa 2:** Paso de parametros.
- **Programa 3:** Sincronización con sleep.

*Solución del primer programa.*

Clase RelojHilosObserver.java

```
public class RelojHilosObserver extends Observable implements Runnable {

    int horas, min, seg;

    public RelojHilosObserver(int horas, int min, int seg) {
        this.horas = horas;
        this.min = min;
        this.seg = seg;
    }

    @Override
    public void run() {
        String t;
        //horas
        while (true) {
```

```

t = "";//para colocar los : en cada separacion y de inicio en poner cero
//horas
if (horas < 10) {
    t += "0" + horas;
} else {
    t += horas;
}
t += ":";
//minutos
if (min < 10) {
    t += "0" + min;
} else {
    t += min;
}
t += ":";
//segundos
if (seg < 10) {
    t += "0" + seg;
} else {
    t += seg;
}
seg++;
if (seg == 60) {
    min++;
    seg = 0;
    if (min == 60) {
        min = 0;
        horas++;
        if (horas == 24) {
            horas = 0;
        }
    }
}
try {
    Thread.sleep(1000);
} catch (InterruptedException ex) {
    System.out.println("Error dormir hilo"+ex.getMessage());
}
//colocamos observable
this.setChanged();
this.notifyObservers(t);
this.clearChanged();
}
}
}

```

Clase relojhilos.java (main)

```
public class relojhilos extends javax.swing.JFrame implements Observer{  
    private void btnIniciarActionPerformed(java.awt.event.ActionEvent evt) {  
        // TODO add your handling code here:  
        this.btnIniciar.setEnabled(false);  
        RelojHilosObserver r = new RelojHilosObserver(10,10,40);  
        r.addObserver(this);  
        Thread t=new Thread(r);  
        t.start();  
    }  
  
    private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {  
        // TODO add your handling code here:  
        System.exit(0);  
    }  
  
    @Override  
    public void update(Observable o, Object arg) {  
        cronometro.setText((String)arg);  
    }  
}
```



Figura 8: Salida del primer programa, Actividad 8

*Solución del segundo programa.*

Clase Hilo1.java

```
public class Hilo1 extends Thread{
    String nombre;
    int edad;

    public Hilo1(String nonmbre, int edad) {

        this.edad = edad;
        this.nombre = nombre;
    }

    @Override
    public void run(){
        System.out.println(nombre + " tiene " + edad + "años");
    }
}
```

Clase Hilo2.java

```
public class Hilo2 extends Thread{
    String nombre;
    int edad;

    @Override
    public void run(){
        System.out.println(nombre + " tiene " + edad + "años");
    }

    public void datos(String nombre, int edad){
        this.edad = edad;
        this.nombre = nombre;
    }
}
```

Clase PasodeParametrosHilos.java (main)

```
public class PasodeParametrosHilos {

    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

```

Hilo1 h1 = new Hilo1("Sofia",22);//Paso parametros con constructor
Hilo2 h2 = new Hilo2();
h2.datos("Ivan",25);//paso parametros con metodo

h1.start();
h2.start();
}

}

```

The screenshot shows the Eclipse IDE's Output window. The title bar says "Output" and "EjercicioReloj (run) x PasoParametrosHilos (run) x". The main pane contains the following text:

```

run:
Sofia tiene 22años
Ivan tiene 25años
BUILD SUCCESSFUL (total time: 0 seconds)

```

Figura 9: Salida del segundo programa, Actividad 8

*Solución del tercer programa.*

Clase Hilo1.java

```

public class Hilo1 extends Thread {

    @Override
    public void run() {
        for (int j = 0; j < 3; j++) {
            System.out.print("H");

            try {
                Thread.sleep(1000);
            } catch (InterruptedException ex) {
                System.out.println("Error sleep"+ ex.getMessage());
            }
        }
    }
}

```

Clase Hilo2.java

```
public class Hilo2 extends Thread {  
  
    @Override  
    public void run() {  
        for (int j = 0; j < 3; j++) {  
            System.out.println(" i");  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException ex) {  
                System.out.println("Error sleep"+ ex.getMessage());  
            }  
        }  
    }  
}
```

Clase Sincronizacionsleepilos.java (main)

```
public class Sincronizacionsleepilos {  
  
    public static void main(String[] args) {  
        // TODO code application logic here  
        Hilo1 h1=new Hilo1();  
        Hilo2 h2=new Hilo2();  
  
        h1.start();  
  
        try {  
            h1.sleep(50);  
        } catch (InterruptedException ex) {  
            System.out.println("Error sleep"+ ex.getMessage());  
        }  
  
        h2.start();  
        try {  
            h2.sleep(50);  
        } catch (InterruptedException ex) {  
            System.out.println("Error sleep"+ ex.getMessage());  
        }  
    }  
}
```

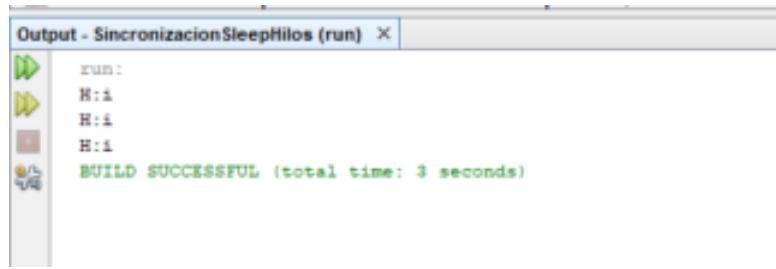


Figura 10: Salida del tercer programa, Actividad 8

### 5.5. Actividad 9 (Tarea)

**Desarrolla un programa GUI carrera de canicas con Observable, dos hilos mínimo.**

*Solución de la tarea.*

Clase Hilo1.java

```
public class Hilo1 extends Observable implements Runnable{
    int x, i;

    @Override
    public void run() {
        x = (int)(Math.random()*400);
        for (i = 490; i > x; i--) {
            try {
                System.out.println(i);
                sleep(7);
            } catch (InterruptedException ex) {
                Logger.getLogger(Hilo1.class.getName()).log(Level.SEVERE, null, ex);
            }
            this.setChanged();
            this.notifyObservers(i);
            this.clearChanged();
        }
    }
}
```

### Clase Hilo2.java

```
public class Hilo2 extends Observable implements Runnable{
    int x, i;

    @Override
    public void run() {
        x = (int)(Math.random()*400);
        for (i = 510; i > x; i--) {
            try {
                System.out.println(i);
                sleep(7);
            } catch (InterruptedException ex) {
                Logger.getLogger(Hilo1.class.getName()).log(Level.SEVERE, null, ex);
            }
            this.setChanged();
            this.notifyObservers(i);
            this.clearChanged();
        }
    }
}
```

### Clase Carrera.java (main)

```
public class Carrera extends javax.swing.JFrame implements Observer{
    Hilo1 h1 = new Hilo1();
    Hilo2 h2 = new Hilo2();

    public Carrera() {
        initComponents();
        this.setLocationRelativeTo(null);
    }

    private void salirActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        System.exit(0);
    }

    private void iniciarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        Thread t = new Thread(h1);
        t.start();
        h1.addObserver(this);

        Thread t2 = new Thread(h2);
    }
}
```

```
t2.start();
h2.addObserver(this);

}

@Override
public void update(Observable o, Object arg) {
    carro1.setLocation(100, (int) arg);
    carro2.setLocation(190, (int) h1.i);
}
}
```



Figura 11: Salida del programa, Actividad 9 (Tarea)

## 5.6. Actividad 10 en clase

Desarrolla los siguientes programas en clase.

- **Programa 1:** Buscador de palabras con hilos.
- **Programa 2:** Sincronización métodos con synchronized Hilos.

*Solución del primer programa.*

Clase Buscador.java

```
public class Buscador extends Thread {  
    String palabra, dir, lectura;  
    int cont=0;  
  
    public Buscador(String palabra, String dir) {  
        this.palabra = palabra;  
        this.dir = dir;  
    }  
  
    @Override  
    public void run(){  
        try {  
            BufferedReader b1 = new BufferedReader(new FileReader(dir));  
            try {  
                while ((lectura = b1.readLine())!=null) {  
                    if (lectura.contains(palabra)) {  
                        cont++;  
                    }  
                }  
                System.out.println("La palabra "+palabra+", se repite "+cont+" veces");  
            } catch (IOException ex) {  
                System.out.println("Error en lectura "+ex.getMessage());  
            }  
        } catch (FileNotFoundException ex) {  
            System.out.println("Error en archivo "+ex.getMessage());  
        }  
    }  
}
```

Clase BuscadorPalabrasHilos.java (main)

```
public class BuscadorPalabrasHilos {  
  
    public static void main(String[] args) {  
        // TODO code application logic here  
        Buscador b1 = new Buscador("data", "C:\\\\Users\\\\karen\\\\Desktop\\\\ejemplo.txt");  
        b1.start();  
    }  
  
}
```

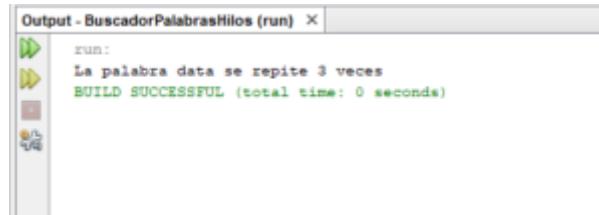


Figura 12: Salida del primer programa, Actividad 10

*Solución del segundo programa.*

Clase Adquiere.java

```
public class Adquiere extends Thread {  
  
    private Control pila;  
    private int n, dormir;  
  
    public Adquiere(Control pila, int n, int dormir) {  
        this.pila = pila;  
        this.n = n;  
        this.dormir = dormir;  
    }  
  
    @Override  
    public void run() {  
  
        try {  
            char c;  
  
            for (int i = 0; i < n; i++) {  
                c = pila.retirar();  
            }  
        } catch (Exception e) {}  
    }  
}
```

```

        System.out.println("Envió: " + c);
        sleep((int) (Math.random() * dormir));
    }
} catch (Exception ex) {
    System.out.println("");
}
}
}
}
```

### Clase Control.java

```

public class Control {

    private char[] pila = null;
    private int tope = 0;
    private boolean lleno = false;
    private boolean vacio = true;

    public Control(int capacidad) {
        pila = new char[capacidad];
    }

    public synchronized void agregar(char c) throws Exception {
        while (lleno) {
            wait();
        }
        pila[tope++] = c;
        vacio = false;
        lleno = tope >= pila.length;
        notifyAll();
    }

    public synchronized char retirar() throws Exception {
        while (vacio) {
            wait();
        }
        char c = pila[--tope];
        lleno = false;
        vacio = tope <= 0;
        notifyAll();
        return c;
    }

}
```

### Clase Producir.java

```
public class Producir extends Thread{  
  
    private Control pila;  
    private int np, dormir;  
  
    public Producir(Control monitor, int np, int dormir) {  
        this.pila = monitor;  
        this.np = np;  
        this.dormir = dormir;  
    }  
  
    @Override  
    public void run(){  
        char c;  
        for (int i = 0; i < np; i++) {  
            c=(char)('A'+i);  
            try {  
                pila.agregar(c);  
                System.out.println("Ingresó: "+c);  
                sleep((int)(Math.random()*dormir));  
            } catch (Exception ex) {  
                System.out.println("Hilo producir"+ex.getMessage());  
            }  
        }  
    }  
}
```

### Clase SincronizacionHilosDiplomado.java (main)

```
public class SincronizacionHilosDiplomado {  
  
    public static void main(String[] args) {  
        // TODO code application logic here  
        Control c = new Control(3);  
        Producir p = new Producir(c, 6, 1500);  
        Adquiere a = new Adquiere(c, 6, 2000);  
        p.start();  
        a.start();  
    }  
}
```

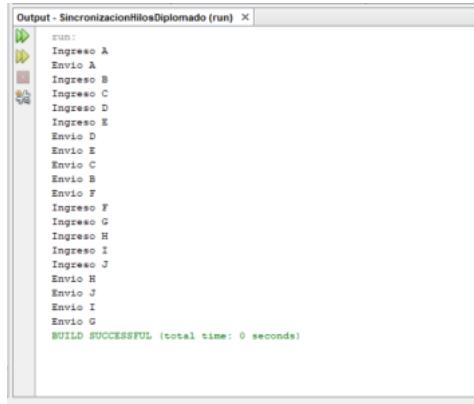


Figura 13: Salida del segundo programa, Actividad 10

### 5.7. Actividad 11 en clase

**Desarrolla un programa Cliente-Servidor utilizando sockets.**

*Solución del programa.*

Clase Cliente.java (main)

```
public class Cliente extends javax.swing.JFrame implements Runnable {

    public Cliente() {
        initComponents();
        Thread hilo2 = new Thread(this);
        hilo2.start();
    }

    private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        System.exit(0);
    }

    private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {
        try {

            Socket sC = new Socket("localhost", 9999);
            ObjectOutputStream salidaMensaje = new ObjectOutputStream(
sC.getOutputStream()
);
            Datos dat = new Datos();
            dat.setIp(ipDestino.getText());
        }
    }
}
```

```

        dat.setMensaje(Mensaje.getText());
        dat.setNombre(nombre.getText());
        salidaMensaje.writeObject(dat);
        AreaTexto.append(Mensaje.getText());
        salidaMensaje.close();

    } catch (IOException ex) {
        System.out.println("Error en la conexión de cliente" + ex.getMessage());
    }
}

//Encapsular los datos del cliente para enviar a servidor
public class Datos implements Serializable {

    private String nombre;
    private String ip;
    private String mensaje;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

    public String getMensaje() {
        return mensaje;
    }

    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }
}

@Override
public void run() {
    int i = 0;
    try {
        ServerSocket CEscucha = new ServerSocket(9999);

```

```

        Socket cliente;
        Datos dat = new Datos();
        while (i == 0) {
            cliente = CEscucha.accept();
            ObjectInputStream flujoEntrada = new ObjectInputStream(
cliente.getInputStream()
);
            dat = (Datos) flujoEntrada.readObject();
        }
    } catch (IOException ex) {
        Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
    }
}
}

```

### Clase Servidor.java (main)

```

public class Servidor extends javax.swing.JFrame implements Runnable {

    public Servidor() {
        Thread hiloServer = new Thread(this);
        hiloServer.start();
        initComponents();
    }

    @Override
    public void run() {
        try {
            ServerSocket ss = new ServerSocket(9999);
            int i = 0;
            while (i == 0) {
                Socket sC = ss.accept();/*
                DataInputStream mensajeEntrada = new DataInputStream(
sC.getInputStream()
);
                String datoTexto = mensajeEntrada.readUTF();
                System.out.println("Texto Servidor " + datoTexto);
                areaTexto.append(datoTexto+"\n");*/
            }
            ObjectInputStream salidaMensaje = new ObjectInputStream(

```

```
sC.getInputStream()
);
        Datos dat = (Datos) salidaMensaje.readObject();
        System.out.println("Texto Servidor " + dat);
        String name = dat.getNombre();
        String ip = dat.getIp();
        String mensaje = dat.getMensaje();
        System.out.println(name + ip + mensaje);

        TextArea.append(name + ":" + ip + "\n" + mensaje);

        Socket enviaDestinatario = new Socket(ip, 9999);
        ObjectOutputStream paqueteReenvio = new ObjectOutputStream
(enviaDestinatario.getOutputStream());
        paqueteReenvio.writeObject(dat);
        enviaDestinatario.close();
        sC.close();

    }

    ss.close();
} catch (IOException ex) {
    Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
} catch (ClassNotFoundException ex) {
    Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
}
}

}
```