



**Ikaskuntza Birtual eta Digitalizatuen LHII**  
CIFP de Aprendizajes Virtuales y Digitalizados

## Tarea Evaluativa 03

**Joseba Sánchez Romero**  
**Programación de servicios y procesos DAM**  
Curso: 2021/22

**EUSKO JAURLARITZA**



**GOBIERNO VASCO**

HEZKUNTZA SAILA  
Lanbide Heziketako Sailburuordetza

DEPARTAMENTO DE EDUCACIÓN  
Viceconsejería de Formación Profesional



## ÍNDICE

1. INTRODUCCIÓN.....	1
2. DIFERENTES APARTADOS DE LA TAREA.....	1
3. AUTOEVALUACIÓN.....	1
4. BIBLIOGRAFÍA .....	1

## 1. INTRODUCCIÓN

## 2. DIFERENTES APARTADOS DE LA TAREA

## 3. AUTOEVALUACIÓN

### Principales problemas encontrados:

#### Tarea 1:

- La posición de la ip address en el array de AddressList (IPAddress ipAddress = ipHostInfo.AddressList[3];). En cada ordenador me aparecía en una posición diferente, hasta que he usado la ip 127.0.0.1 y funciona en todos.
- Los condicionales *if* tienen que ser *else if* para que no se ejecuten todos en el multihilo y solo vaya por la condición que se cumple.
- Que se pase por consola el número 0 como en la explicación de la tarea, lo he hecho igual para hacerlo como el vídeo pero no sé por qué.

```
73
Has acertado!!Zorionak!
45
0
GANADOR
Identificador: 1000
Cliente: 0
Has acertado!!Zorionak!
```

- Asegurar con un booleano que solo entre una vez para escribir el nombre del ganador en el servidor cada vez que un cliente sigue metiendo números. Si no, lo escribe un par de veces.

#### Tarea 2:

- Mostrar al cliente la ruta donde ha guardado el fichero:  
(System.IO.FileStream)fileStream).Name



## Funcionalidad:

### Tarea 1:

<https://www.youtube.com/watch?v=aUXq1vWsg3U>

## Lado Servidor

1. Inicializar variables con datos de conexión y llamada a servidor.juegoLoteria para crear Socket listener y definir número al azar.

```
0 referencias
public static int Main(string[] args)
{
    int port = 13000;
    IPAddress localAddr = IPAddress.Parse("127.0.0.1");
    Program servidor = new Program(localAddr, port);

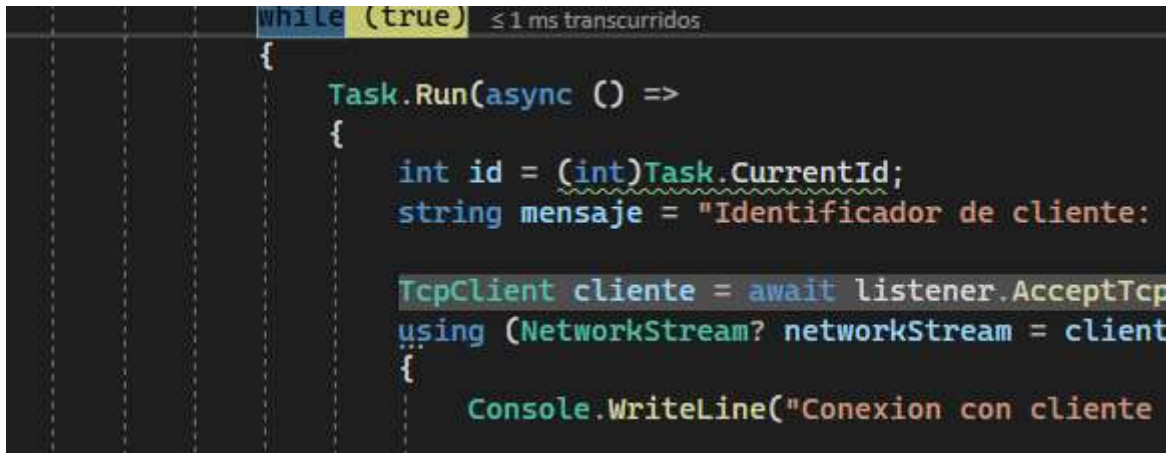
    servidor.juegoLoteria();

    return 0;
}
```

```
int numeroSecreto = rnd.Next(nmin, nmax);
Console.WriteLine("El numero aleatorio es: {0}", numeroSecreto);

listener.Start();
listening = true;
Console.WriteLine("Socket listener creado.");
bool finPartida = false;
int idGanador = 0;
int idPerdedor = 0;
//Console.ReadKey();
object o = new object();
```

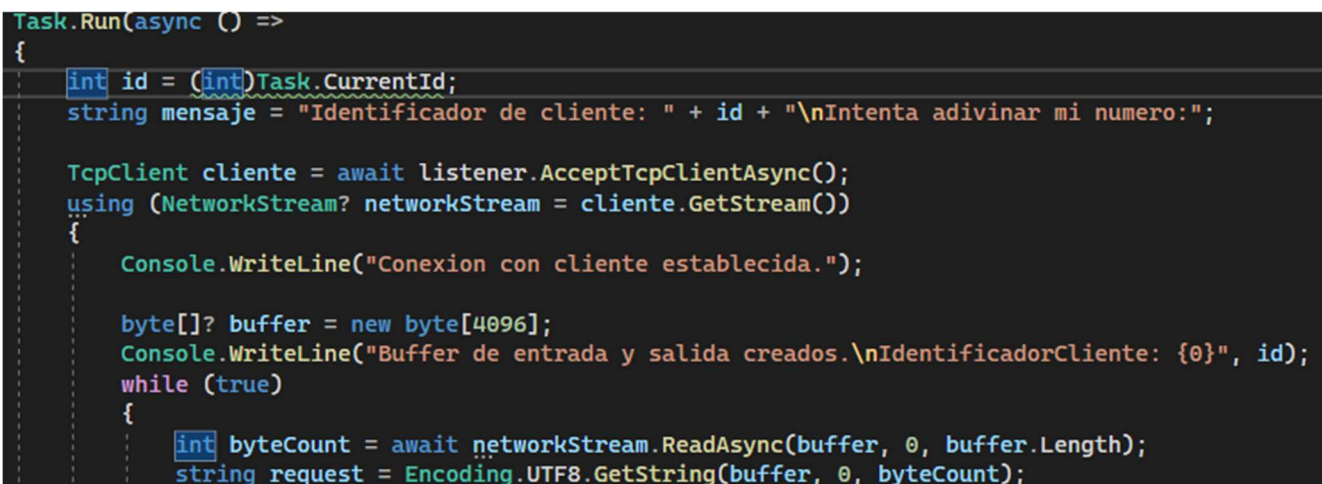
2. Servidor en espera de clientes conectados con while(true).



```
while (true) ≤ 1 ms transcurridos
{
    Task.Run(async () =>
    {
        int id = (int)Task.CurrentId;
        string mensaje = "Identificador de cliente: ";

        TcpClient cliente = await listener.AcceptTcpClientAsync();
        using (NetworkStream? networkStream = cliente.GetStream())
        {
            Console.WriteLine("Conexion con cliente");
        }
    });
}
```

3. Guardar id de la tarea para asignárselo al cliente que se conecte y preparar el texto de bienvenida en el string *mensaje*. Esperar a que se conecte con await listener AcceptTcpClientAsync. Usar la conexión creada con el cliente a partir de ahora mediante using NetworkStream. Preparar buffer de intercambio de mensajes y esperar mensaje de cliente para identificarlo.



```
Task.Run(async () =>
{
    int id = (int)Task.CurrentId;
    string mensaje = "Identificador de cliente: " + id + "\nIntenta adivinar mi numero:";

    TcpClient cliente = await listener.AcceptTcpClientAsync();
    using (NetworkStream? networkStream = cliente.GetStream())
    {
        Console.WriteLine("Conexion con cliente establecida.");

        byte[]? buffer = new byte[4096];
        Console.WriteLine("Buffer de entrada y salida creados.\nIdentificadorCliente: {0}", id);
        while (true)
        {
            int byteCount = await networkStream.ReadAsync(buffer, 0, buffer.Length);
            string request = Encoding.UTF8.GetString(buffer, 0, byteCount);
        }
    }
}
```



4. Batería de condicionales if para identificar en qué momento de la partida se encuentran. Lo primero que enviará el cliente será el string con la palabra "cliente" para identificarle y empezar la partida.

```
if (finPartida == true && id != idGanador && id != idPerdedor)
{
    idPerdedor = id;
    Console.WriteLine(request);
    request = "El ganador es: " + idGanador.ToString();
    Console.WriteLine("0\\nGANADOR\\nIdentificador: {0}\\nCliente: 0\\nHas acertado!!Zorionak!", idGanador);
}
else
{
    if (request.Equals("cliente"))
    {
        request = id.ToString();
    }
    else if (int.Parse(request) > numeroSecreto)
    {
        Console.WriteLine(request);
        request = "El numero es menor.";
    }
    else if (int.Parse(request) < numeroSecreto)
    {
        Console.WriteLine(request);
        request = "El numero es mayor.";
    }
    else if (int.Parse(request) == numeroSecreto)
    {
        Console.WriteLine(request);
        request = "Has acertado!!Zorionak!";
        Console.WriteLine(request);
        finPartida = true;
        idGanador = id;
    }
}
```



5. Envío mensajes de servidor a cliente después de pasar por los condicionales y cierre de listener cuando haya ganador.

```
byte[] ServerResponseBytes = Encoding.UTF8.GetBytes(request);  
await networkStream.WriteAsync(ServerResponseBytes, 0, ServerResponseBytes.Length);  
}  
}  
});  
}  
}  
  
catch (Exception e)  
{  
    Console.WriteLine("Exception: {0}", e);  
}  
finally  
{  
    listening = false;  
}
```



## Lado Cliente

1. Archivo *Cliente1.cs* donde se encuentran todas las funciones que se llaman desde *Program.cs* para hacer las conexiones en respuesta a lo que hemos visto en el lado Servidor.

```
1 referencia
public Cliente1()
{
    ipAddress = IPAddress.Parse("127.0.0.1");
    sender = new Socket(ipAddress.AddressFamily,
        SocketType.Stream, ProtocolType.Tcp);
    Console.WriteLine("Socket Cliente creado.");
}

1 referencia
public void establecerConexion()
{
    IPEndPoint remoteEP = new IPEndPoint(ipAddress, port);
    sender.Connect(remoteEP);
    Console.WriteLine("Buffer de escritura y lectura creados.");
}

2 referencias
public void transfiendoInfo(string datos)
{
    byte[] msg = Encoding.ASCII.GetBytes(datos);
    int bytesSnd = sender.Send(msg);
}

2 referencias
public string recibiendoInfo()
{
    object o = new object();
    lock (o)
    {
        byte[] bytes = new byte[1024];
        int bytesRec = sender.Receive(bytes);
        string datos = Encoding.ASCII.GetString(bytes, 0, bytesRec);
        return datos;
    }
}
```





2. Archivo *Program.cs* donde se definen todas las variables a utilizar y se hacen las llamadas a las funciones guardadas en *Cliente1.cs* para hacer las conexiones.

```
0 referencias
public static int Main(string[] args)
{
    IPEndPoint ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
    IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
    Random rdm = new Random();
    int nmax = 101;
    int nmin = 1;

    try
    {
        Cliente1 cliente = new Cliente1();
        cliente.establecerConexion();
        cliente.transfiriendoInfo("cliente");
        string msg = cliente.recibiendoInfo();
        Console.WriteLine("Identificador de cliente: " + msg.ToString() + "\nIntenta adivinar mi numero:");
        int njugadas = 0;
        string findelapartida = "*****\nFin de la partida.\n*****\nFin del juego";
        while (true)
        {
```

3. Elección de número al azar para proponer al Servidor según las pistas que nos ha dado anteriormente y todos los condicionales preparados para el intercambio de mensajes entre Servidor y Cliente, igual que hemos visto en lado Servidor.

```
while (true)
{
    //Thread.Sleep(1000);
    string cadena = rdm.Next(nmin, nmax).ToString();
    cliente.transfiriendoInfo(cadena);
    Console.WriteLine(cadena);
    njugadas++;
    msg = cliente.recibiendoInfo();
    Console.WriteLine(msg);
    if (msg.Equals("Has acertado!!Zorionak!") || cadena.Equals("salir"))
    {
        Console.WriteLine("Numero de jugadas realizadas por ti: {0}", njugadas);
        Console.WriteLine(findelapartida);
        break;
    }
    else if (msg.Substring(0, 13).Equals("El ganador es"))
    {
        Console.WriteLine(findelapartida);
        break;
    }
    else if (msg.Equals("El numero es mayor."))
    {
        nmin = int.Parse(cadena);
    }
    else if (msg.Equals("El numero es menor."))
    {
        nmax = int.Parse(cadena);
    }
}
```

#### 4. Cierre de conexiones cuando haya ganador y catches.

```
    }  
    cliente.cerrarCliente();  
}  
catch (SocketException se)  
{  
    Console.WriteLine("Cliente ha tenido problemas de SocketException : {0}", se.ToString());  
}  
catch (Exception ex)  
{  
    Console.WriteLine("Cliente ha tenido problemas al establecer la conexión con el servidor");  
    Console.WriteLine(ex.ToString());  
}  
  
Console.WriteLine("\nPresiona intro para continuar...");  
Console.Read();  
  
return 0;  
}
```



## Tarea 2:

<https://www.youtube.com/watch?v=WPc06eZ00To>

### Lado Servidor

1. Apertura de socket TcpListener y socket TcpClient. Variable contador para admitir hasta 5 clientes y denegar acceso al resto. Llamada a *ManipuladorCliente* para atender sus peticiones.

```
0 referencias
static void Main(string[] args)
{
    TcpListener serverSocket = new TcpListener(IPAddress.Any, 13000);
    TcpClient clientSocket = new TcpClient();
    int counter = 0;
    serverSocket.Start();
    Console.WriteLine("Servidor iniciado en. . . " + serverSocket.LocalEndpoint);
    counter = 0;

    while (true)
    {
        if (counter >= 5) { Console.WriteLine("Numero maximo de clientes conectados, acceso denegado"); return; }
        counter += 1;
        clientSocket = serverSocket.AcceptTcpClient();
        Console.WriteLine(">> Cliente numero " + counter + " : " + clientSocket.Client.AddressFamily);
        ManipuladorCliente manipuladorCliente = new ManipuladorCliente();
        manipuladorCliente.StartCliente(clientSocket, Convert.ToString(counter));
    }
}
```

2. Definir directorio donde guardaremos las fotos a ofrecer. Definir hilo, array bytes y resto de variables para la conexión.

```
2 referencias
public class ManipuladorCliente
{
    private const string Directorio = @"..\..\fotos";
    TcpClient _clientSocket;
    string _clientNo = "";
    private Thread _clientThread;
    private byte[] _bytesFrom;
    private NetworkStream _networkStream;
    private TcpListener _serverSocket;

    1 referencia
    public void StartCliente(TcpClient clientSocket, string clientNo)
    {
        this._clientSocket = clientSocket;
        this._clientNo = clientNo;
        _clientThread = new Thread(Manipulador);
        _clientThread.Start();
        _networkStream = clientSocket.GetStream();
    }
}
```



3. Abrir buffer para intercambio de datos, llamada a *ArbolEnvioFichero* para mostrar el árbol con el contenido de los ficheros disponibles y llamada a *ComprobarFicheroDescargado*.

```
1 referencia
private void Manipulador()
{
    bool fileFlag = false;
    while (true)
    {
        if (!fileFlag)
        {
            ArbolEnvioFichero();
            fileFlag = true;
        }
        try
        {
            _bytesFrom = new byte[_clientSocket.ReceiveBufferSize];
            _networkStream.Read(_bytesFrom, 0, (int)_clientSocket.ReceiveBufferSize);
            string dataFromClient = Encoding.ASCII.GetString(_bytesFrom);
            dataFromClient = dataFromClient.Substring(0, dataFromClient.Length);
            ComprobarFicheroDescargado(dataFromClient);
        }
        catch (Exception e)
        {
            Console.WriteLine("Catch desde Servidor >>> " + e.Message);
        }
    }
}
```

4. Función *ComprobarFicheroDescargado* para abrir buffer con el tamaño de la imagen a descargar.

```
1 referencia
private void ComprobarFicheroDescargado(string dataFromClient)
{
    FileRequest fileRequest = JsonConvert.DeserializeObject<FileRequest>(dataFromClient);
    if (fileRequest != null)
    {
        try
        {
            byte[] fichero = File.ReadAllBytes(fileRequest.Path);
            _bytesFrom = new byte[fichero.Length];
            _networkStream.Write(fichero, 0, fichero.Length);
            _networkStream.Flush();
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```



5. Función *GetFilesInfo* que es llamada desde *ArbolFichero* para mostrar los datos aunque solo utilizaremos los nombres definidos en el array string *nombres*.

```
1 referencia
private string GetFilesInfo()
{
    List<FilesInfo> infos = new List<FilesInfo>();
    string[] files = System.IO.Directory.GetFiles(Directorio, "*.*").Select(Path.GetFileName).ToArray();
    string[] nombres = { "1.- FotoCiudad", "2.- FotoMonte", "3.- FotoPlaya" };
    int x = 0;

    foreach (string file in files)
    {
        FileInfo fileInfo = new FileInfo(Path.Combine(Directorio, file));
        infos.Add(new FilesInfo
        {
            FileName = nombres[x],
            AbsolutePath = fileInfo.FullName,
            Extension = fileInfo.Extension,
            DateCreated = fileInfo.CreationTimeUtc,
            Size = fileInfo.Length
        });
        x++;
    }
    string serializeObject = JsonConvert.SerializeObject(infos);

    return serializeObject;
}
```



## Lado Cliente

1. Aplicación de Windows Forms con los típicos archivos para definir la IU y sus funcionalidades. En una de las funciones guardadas en Form1.cs definimos el mensaje para mostrar al usuario después de guardar el archivo con el path de guardado o informando de la cancelación de la descarga.

```
if (downloadEnabled)
{
    inStream = new byte[_client.ReceiveBufferSize];
    int i = _serverStream.Read(inStream, 0, _client.ReceiveBufferSize);

    if (DialogResult.OK == (new Invoker(dialog).Invoke()))
    {
        Stream fileStream = dialog.OpenFile();
        fileStream.Write(inStream, 0, i);
        string fs = ((System.IO.FileStream)fileStream).Name;

        fileStream.Close();
        downloadEnabled = false;
        MessageBox.Show("Archivo descargado en " + fs);
    }
    else
    {
        MessageBox.Show("Descarga cancelada.");
    }
}
```

## 4. BIBLIOGRAFÍA

